1. Typologie der Datenbanksysteme

1.1 Grenzen der herkömmlichen Datenverarbeitung

Es heißt: Datenbanksysteme überwinden die Grenzen herkömmlicher Datenverarbeitung. Das kann sich nur auf das Bearbeiten von Dateien mit Dateisystemen beziehen. Das zeigt bspw. folgende 1. Aufgabe, die sehr vereinfacht eine maschinelle Gehaltsabrechnung beschreibt:

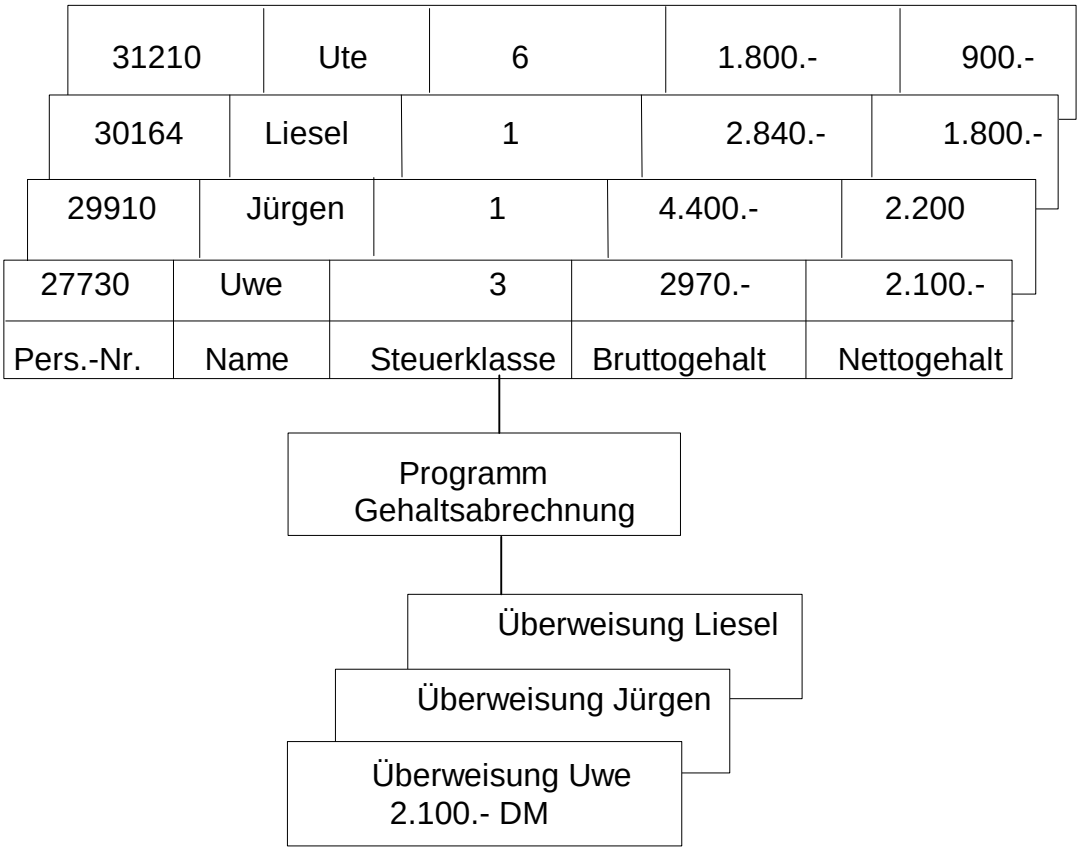


Abb. 1.1-1: Datenfluß zur 1.Aufgabe

Das Programm liest einen Datensatz in einen festen Bereich des Hauptspeichers ein und baut die Druckausgabe auf. Diese Druckausgabe besteht aus Überweisungen auf das Konto des Mitarbeiters bei seiner Bank.

Eine 2. Aufgabe ist: Beschriftung von Aufklebern mit den Adressen der Mitarbeiter (z.B. für den Versand von Mitteillungen).
Es gibt zwei Möglichkeiten, diesem Programm die Datenfelder "Name" und "Adresse" zur Verfügung zu stellen:

Lösung 1: Erweiterung des bestehenden Datensatzes für die Gehaltsabrechnung um ein Adressenfeld

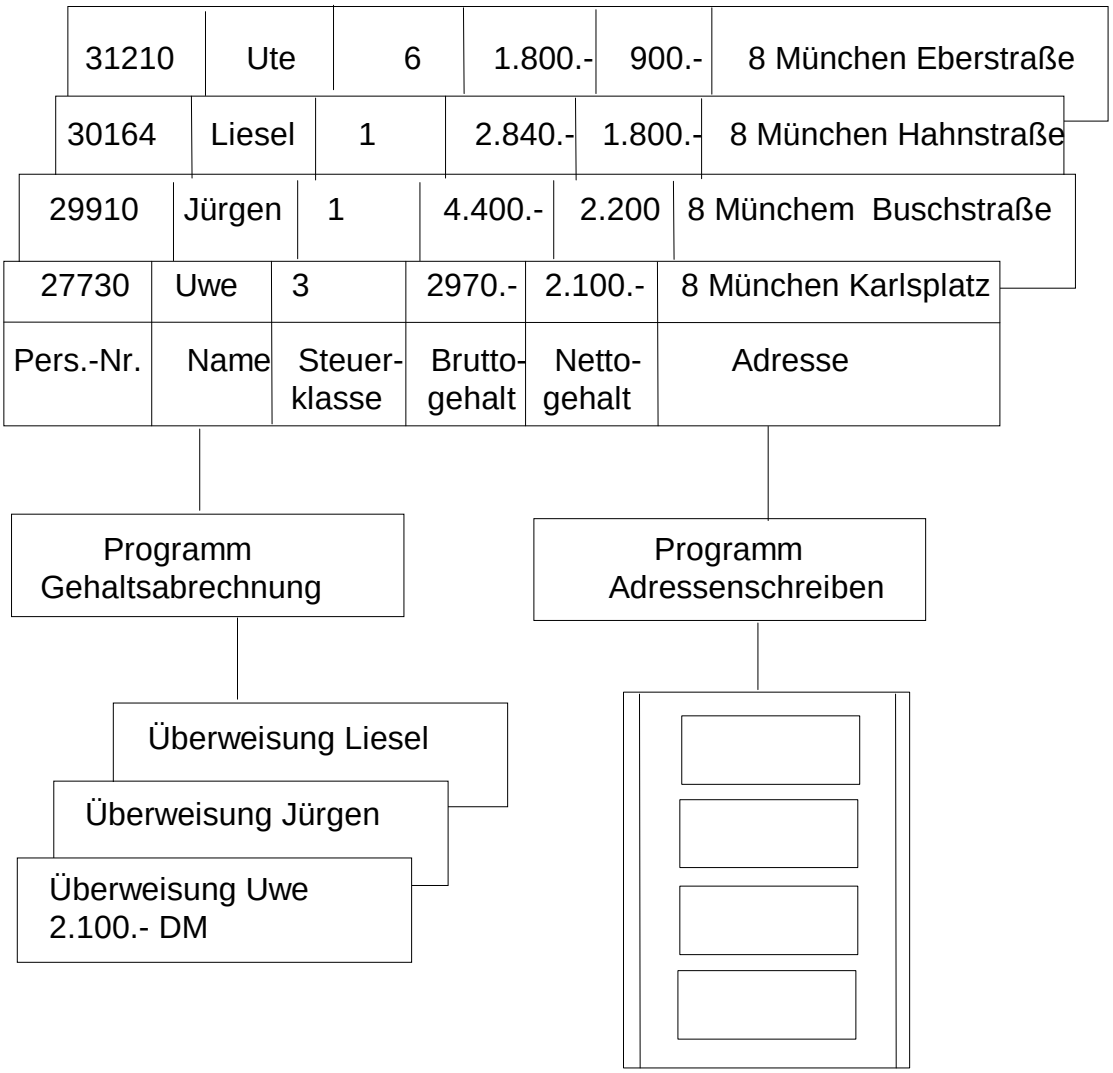


Abb. 1.1-2: Datenfluß zur 2. Aufgabe

Aus dieser Lösung ergeben sich zwei Schwierigkeiten:

1. Der Datensatz vergrössert sich. Das bestehende Programm Gehaltsabrechnung muß geändert werden, denn der Eingabebereich muß entsprechend vergrössert werden.Erweiterung von Datensätzen bedeutet: Änderung aller bestehenden Programme, die mit den Datensätzen arbeiten.
2. Das Programm, das die Aufkleber mit Adressen beschriftet, muß den erweiterten Datensatz einlesen. Damit kann man in diesem Programm auch alle anderen Personaldaten verarbeiten. Die Daten sind nicht mehr vertraulich (Datenschutz!).

Lösung 2:

- Der Datenbestand für die Gehaltsabrechnung bleibt unverändert. Damit entfallen Programmänderungen
- Der Datenbestand für die Adressenschreibung wird völlig neu aufgebaut

Abb. 1.1-3: Datenbestand für die Adressenschreibung

Es ergeben sich aus dieser Lösung allerdings zwei neue Probleme: Bestimmte Ordnungsbegriffe sind doppelt zu speichern (Pers.Nr., Name). Man spricht von Datenredundanz (gleiche Informationen sind mehrfach abgespeichert).
Daraus ergibt sich:

1. Zusätzlicher Bedarf an Speicherplatz auf externen Speichereinheiten
2. Unterschiedlicher Stand der Datenbestände, wenn nicht alle Änderungen in mehrfach gespeicherten Daten durchgeführt werden.

Eine 3. Aufgabe ist:

Für jeden Mitarbeiter sind Informationen über seine Ausbildung bzw. Erfahrungen und die Möglichkeiten seiner Beschäftigung zu speichern. Mitarbeiter Müller kann z.B. im Unternehmen zwei Beschäftigungen ausüben. Für jede Beschäftigung hat er eine entsprechende Ausbildung bzw. Erfahrung.

Es ergeben sich wieder zwei Lösungen für die 3. Aufgabe:

Lösung 1: Erweiterung des gemeinsamen Datensatzes
Lösung 2: Aufbau eines dritten Datenbestandes

Probleme:

Wie viele Beschäftigungen kann ein Mitarbeiter möglicherweise dann ausüben?
Wie viele Ausbildungen bzw. Erfahrungen kann er haben?
Wieviel Speicherplatz muß für neue Ausbildungen bzw. Erfahrungen im Datensatz je Mitarbeiter frei gelassen werden?

Folgerungen

In der herkömmlichen Datenverarbeitung treten eine Reihe von Problemen auf. Diese Probleme verhindern den Aufbau von Informationssystemen. Folgende Forderungen sind daher von einem System zu erfüllen, das diese Probleme lösen soll:

1. Keine Programmänderung bei Erweiterung von Datensätzen
2. Zugriff nur auf solche Daten im Programm, die für die Problemlösung notwendig sind
3. Redundanzfreie Speicherung von Informationen
4. Beliebige Anzahl von Einfügungen solcher Informationen, die mehrfach auftreten können (z.B. mehrere Ausbildungen der Mitarbeiter)

1.2 Erläuterung von Begriffen

1.2.1 Datenbanken und Datenbanksyteme

Die Grenzen herkömmlicher (nicht integrierter) Datenverarbeitung (sog. Insellösungen) sind zu überwinden. Das kann durch eine zentrale Datenhaltung in einem Datenbanksystem erreicht werden. Datenbanken sind in der geschilderten Weise die Verwirklichung des Gedankens der integrierten Datenverarbeitung.

Was versteht man demnach unter einer **Datenbank** bzw. einem **Datenbank-system**?
Eine **Datenbank** (DB) ist eine Sammlung von Datenbeständen in einer hochgradig integrierten Speicherung (Speicher mit Direktzugriff) mit vielfältigen Verarbeitungsmöglichkeiten.
Ein **Datenbanksystem** (DBS) ist ein Software-Paket, das

- neutral große Datenmengen verwaltet
- die Beziehungen der möglichst redundanzfreien Daten zueinander kontrolliert
- unterschiedliche Verarbeitungs- und Auswertungsmöglichkeiten der Daten gestattet.

Das bedeutet: Alles, was oft als Datenbank (z.B. Oracle, Access, dBASE) angeboten wird, sind Datenbanksysteme (**DBS**). Das sind Programme, mit deren Hilfe Datenbanken für interne Zwecke aufgebaut und gepflegt werden können. Es handelt sich dabei um Datenbankverwaltungs-Systeme (Datenbankmanagement-Systeme, DBMS), die nur die Organisation, nicht aber den Inhalt der Datenbank bestimmen.
Streng genommen bezeichnet der Begiff „Datenbank“ eine nach bestimmten Regeln aufgebaute Datensammlung (Texte, Tabellen bzw. Zahlen, Zeichen, Graphiken). Auf diese Daten besteht eine Zugriffsmöglichkeit, die über vielfältige Kombinationsmöglichkeiten des Suchbegriffs gewünschte Informationen bereitstellt. So wird heute weltweit der Zugriff auf mehr als 6000 öffentlich zugängliche Datenbanken angeboten. Sie werden von zahlreichen Instituten, Zeitschriften, Zeitungen, Nachrichtenagenturen, Vorlagen und amtlichen Stellen gefüllt. In **externen Datenbanken** arbeitet der Anbieter einer Datenbank mit Hilfe von Datenbanksystemen zusammen. Über Datenbanksysteme werden externe Quellen (Zeitschriften, Börsenmitteilungen, Gesetze, Urteile, etc.) ausgewertet und in eine für die Datenbank geeignete Form gebracht. Im Gegensatz dazu enthalten **interne Datenbanken** nur Daten, die im eigenen Umfeld des Datenbankadministrators anfallen.

1.2.2 Informationssysteme

Was gehört alles zu einem **Informationssystem**?

Eine Datenbank[1] liefert **Fakten** für ein Informationssystem. Ausgangspunkt solcher Systeme ist der Benutzer. Er stellt Fragen, das Informationssystem gibt Antworten. Einfache Anfragen können direkt aus der Datenbank beantwortet werden.

Bsp.:

Datenbank	Frage
Buchhaltung	Letzjähriger Gesamtumsatz
Lexikon	Geburtsjahr von J. S. Bach

Für komplizierte Fragen sind zusätzlich Methoden zur Kombination bestimmter Daten aus der Datenbank nötig
Methodenbanken sind Programme für mathematische Verfahren (Matrizen-, Differential- und Integralrechnung), statistische Auswertungen und Operations Research. Methodenbanken sind auf Großrechern (Mainframe) immer noch selten. Häufig anzutreffen dagegen sind integrierte Programmpakete für Mikrocomputer, die Programme zur Datenbankanwendung sowie für Präsentationsgrafik, Tabellen-kalkulation und Textverarbeitung enthalten.
Damit ist die Basis geschaffen für weitere Nachforschungen, die mehr als Faktenermittlung und zugehörige spezifische Auswertung umfassen.

Bsp.:

Methodenbank	Datenbank	Frage
Betriebsabrechnung	Buchhaltung	Vollkosten der Maschine X je Stunde
Prognose-Methoden	Bevölkerungsdaten	Schulanfänger in 10 Jahren

Manchmal sind weitere Nachforschungen nötig.

Bsp.:

Datenbank	zusätzliche Nachforschungen	Frage
Buchhaltung	Geschäftsberichte der Konkurrenz	Personalkosten im Vergleich zur Branche

In speziellen Anwendungen muß der Datenbestand für den Fragesteller durch eine
„Organisation der Datenverarbeitung“ aufbereitet werden. Methoden, Recherchen (Nachforschungen) und Speicherung der Daten bilden durch diese „Organisation der Datenverarbeitung“ ein Informationssystem.

[1] vgl. 1.2.1

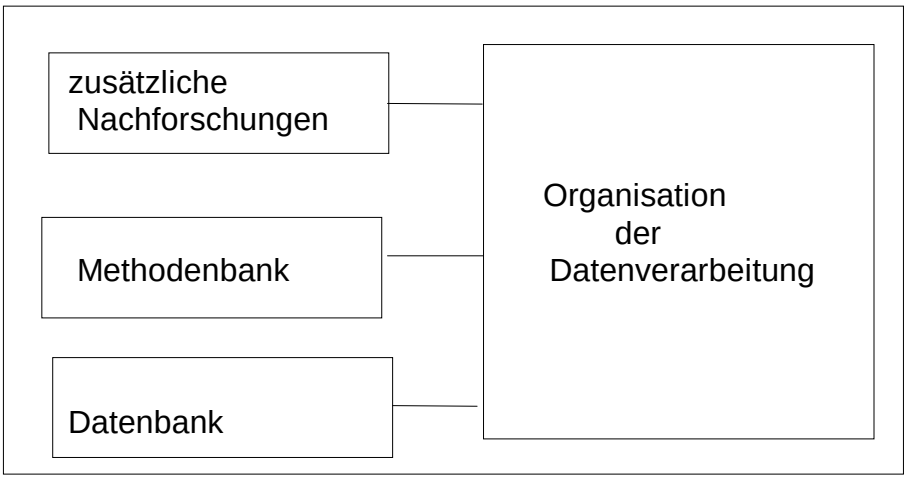


Abb. 1.2-1: Aufbau eines Informationssystems

Die „Organisation der Datenverarbeitung“ ist weitgehend bestimmt durch die Anwendung. Solche Organisationsformen haben sich herausgebildet für Dokumentationssysteme, Expertensysteme, „Data Warehouse“-Systeme.

1.2.2.1 Dokumentations-Systeme

Informations- bzw. Dokumentations-Systeme enthalten:

Stichwortkataloge

Die Titel oder die Zusammenfassungen der zu dokumentierenden Zeitschriftenartikel oder Bücher werden auf aussagenkräftige Begriffe durchsucht. Weggelassen werden dabei alle Wörter, die nichts zum Suchprozeß beitragen (Artikel, Präpositionen, Konjunktionen, wenig aussagekräftige Wörter).

Schlagwortkataloge

Nicht aus dem ursprünglichen Dokument bestimmte Stichwörter sondern speziell ausgewählte Schlagwörter aus einem Schlagwortregister bilden hier die Grundlage. Der Benutzer muß seine Abfrage mit diesen Schlagwörtern (Deskriptoren) aufbauen (vgl. Schlagwortverzeichnis der Association of Computing Machinery (ACM)).
Deskriptoren können aber zu Problemen der folgenden Art führen:

- das Synonymenproblem (Mehrfachbenennung eines Begriffs)
- das Polysemenproblem (Verwendung einer Begriffsbenennung für verschiedene Begriffe)
- das Äquivalenzproblem

Verwandte Deskriptoren werden zweckmäßigerweise zu Äquivalenzklassen zusammengefaßt. Damit kann eine Dokumentensuche u.a.a. über einen zunächst

gegebenen Deskriptor erweitert werden. Das Problem dabei ist, inwieweit die zu einer Äquivalenzklasse zusammengefaßten Deskriptoren wirklich "gleich" sind.

Thesaurus

Ein **Thesaurus** (Wortschatz) ist ein Verzeichnis von verschiedenen Schlagwörtern.
Thesauri werden verwendet:

- als vordefinierte Deskriptorenliste
- als Synonymenwörterbuch
- zur Gruppierung, Klassifizierung oder Strukturierung von Deskriptoren

In verschiedenen Fachgebieten werden umfassende Thesauri angeboten.

Bsp.: "Engineers Joint Council Thesaurus"
Ein Auszug aus diesem Thesaurus enthält Vorzugsbenennungen und zusätzliche Begriffsbestimmungen, die in Beziehung zu den Vorzugsbenennungen gesetzt sind. Die Beziehungen sind gekennzeichnet durch:

UF (used for)
Die vorstehende Vorzugsbenennung ist synonym mit der durch UF gekenn-zeichneten Begriffsbenennung.
BT (broader term)
Die mit BF gekennzeichnete Begriffsbenennung umfaßt die vorstehende Vor-zugsbenennung in ihrer Bedeutung.
NT (narrower term)
Die mit NT gekennzeichnete Begriffsbenennung ist hins. ihrer Bedeutung in der vorstehenden Begriffsbenennung enthalten.
USE
Die mit USE gekennzeichnete Begriffsbenennung ist als synonyme Vor-zugsbenennung für die vorstehende Begriffsbenennung zu verwenden.
RT (related term)
Die mit RT gekennzeichnete Begriffsbenennung ist in ihrer Bedeutung mit der vorstehenden Vorzugsbenennung verwandt (, jedoch nicht im Sinne einer BT- oder NT-Beziehung).

Vorzugsbenennungen sind unterstrichen, z.B.:

```
Koaxialkabel
     UF Koaxialleitung
     NT Flüssigkeitsgefüllte Koaxialkabel
     BT Übertragungskabel
     RT Starkstromkabel

Koaxialleitung
     USE Koaxialkabel

Koaxialfilter
     BT Elektr. Filter
     RT Mikrowellenfilter
Kobalt
     BT Metall
```

Ein **Thesaurus** ist eine Dokumentationssprache[2], die aus Deskriptoren zur eindeutigen Begriffsbenennung (Vorzugsbenennung) und zusätzlichen Wörtern der natürlichen Sprache (ergänzende Begiffsbenennung, Hilfsmittel zur Darstellung von Beziehungen zwischen diesen Benennungen) besteht. Vorzugsbenennungen sind für die Informationswiedergewinnung die entscheidenden Größen. Unter ihnen gibt es keine Polyseme oder Synonyme. Vorzugsbenennungen sind in der Regel durch zusätzliche Begriffsbestimmungen ergänzt, damit eine vielschichtige Beschreibung der Dokumente möglich ist. Eine weitere zentrale Aufgabe in einem Thesaurus ist neben der Bestimmung von Vorzugsbenennungen und der darüber hinaus erlaubten Begriffsbenennungen die Festlegung von Beziehungen zwischen den Begriffsbenennungen.
Folgende Beziehungen findet man in Thesauri am häufigsten: „übergeordneter Begriff (OB), untergeordneter Begriff (UB), verwandter Begriff (VB)“. Wenn dann A bspw. Oberbegriff von B ist, ist zugleich B Unterbegriff von A. Wenn A mit B verwandt ist, ist auch B mit A verwandt. Die Beziehung (Relation) „verwandter Begriff“ ist damit symmetrisch.

Systematischer Katalog

Darunter verstehen Bibliothekare die Verwendung eines künstlichen Schlagwortverzeichnisses, nämlich der "universellen Dezimal-Klassifikation (UDK oder DK)".

Dewey's Dezimalklassifikation
Um 1870 entwickelte der amerikanische Bibliothekar Melvil Dewey ein Ordnungssystem für eine einheitliche Bücheraufstellung. Er teilte das gesamte "menschl. Wissen" in 10 Hauptabteilungen:

```
0  Allgemeine Medizin
1  Philosophie
2  Religion, Theologie
3  Sozialwissenschaften, Recht, Verwaltung
4  Sprachen
5  Mathematik
6  Technik, Medizin
7  Kunst
8  Literatur
9  Geschichte
```

Jeder der Hauptabteilungen teilt sich in 10 Abteilungen (00, .. ,09, 10, .. ,19,20 ...) und bei Bedarf jede dieser Abteilungen in 10 Unterabteilungen (000, 001, ...). Dieser Klassifikationsvorschlag fand eine sehr starke Verbreitung. Die verwendete Dokumentationssprache ist einfach zu lernen, denn sie besteht ja nur aus etwa 1000 Deskriptoren. Die Deskriptoren sind hierarchisch geordnet.
Die internationale Dezimalklassifikation
Das Klassifikationssystem von M. Dewey wurde bis heute immer wieder weiter überarbeitet. Durch weitere Untergliederung entstanden bereits bis 1973 70000 Deskriptoren. Die internationale Dezimalklassifikation wird heute gepflegt von der "Federation Internationale de Dokumentation". Diese Kataloge bilden die

[2] Regeln für die Erstellung und Weiterentwicklung von Thesauri sind in DIN 1463 festgelegt

Hilfsorganisation für Suchfragen. Sie erlauben dem Dokumentationsbenutzer einen schnelleren Zugang zur gesuchten Information.

1.2.2.2 Data Warehouse

Ein Data Warehouse[3] ist eine Datenbank mit historischen Daten, die aus unterschiedlichen Datenquellen eines Unternehmens in eine seperate Datenbank kopiert werden. Herkömmliche Datenbanken (operative Systeme) bearbeiten das Tagesgeschäft. Informationen über z.B. die Umsatzsituation, Soll/Ist-Werte einer Budget-Planung sind daraus leicht abzuleiten. Informationen über die Umsatzentwicklung der letzten 6 Jahre, Soll/Ist-Entwicklung im Vergleich zum Vorjahr können nicht direkt ermittelt werden.Ein **Data Warehouse** stellt Datenmaterial bereit, derartige komplexe Fragen zu beantworten. Es enthält Daten (in unterschiedlicher Verdichtung), die für die Entscheidungsunterstützung der Benutzer relevant sind. Direktzugriff wird Endbenutzern durch einen Informationskatalog ermöglicht, der über Inhalte, Formate und Auswertungsmöglichkeiten des Data Warehouse Auskunft gibt.
Die Auswertung dieser Datenbank durch eine große Anzahl von Benutzern stellt große Anforderungen. Der Erfinder des Datenbankmodells für relationale Datenbanken, E.F.Codd, hat folgende Merkmale genannt, die Systeme für das "Online Analytical Processing (OLAP)" erfüllen sollen:

- Merdimensionale, konzeptionelle Sicht auf die Daten
- Transparenz und Integration in die operativen Systeme
- Zugänglichkeit unterschiedlicher Datenbanken über eine logische Gesamtsicht
- stabile, volumenunabhängige Antwortzeiten
- Client-/Server-Architektur
- Mehrbenutzerunterstützung
- flexibles Berichtswesen
- unbeschränkt dimensionsübergreifende Operatoren

1.2.2.3 Expertensysteme

Ein Expertensystem ist ein Programm, das sich in irgendeinem Anwendungs-bereich wie ein Experte verhält. Expertensysteme müssen fähig sein, Probleme zu lösen, die Expertenwissen in einem bestimmten Bereich verlangen. Sie sollen in irgendeiner Form Wissen verarbeiten (wissensbasierte Systeme).
Ein voll ausgebautes Expertensystem besteht im allg. aus 5 verschiedenen Komponenten:

[3] Die Hersteller (z.B. Oracle, Sybase, Software A.G., IBM, Siemens etc.) vermarkten unter diesem Schlagwort alle möglichen Werkzeuge, die mit der Verwaltung, Integration und Auswertung großer Datenbestände zu tun haben.

- Die **Wissensbasis**
  bildet die Grundlage. Sie enthält die Kenntnisse des Experten, meistens in Form von Fakten und Regeln oder auch als Rahmen (Beschreibung von Objekten) und Skripten (Beschreibung von Abläufen).
- Die **Inferenz-Maschine** (inference machine)
  dient der Wissensauswertung. Sie sucht und verknüpft Fakten und Regeln nach einer vorgegebenen Strategie und produziert Forderungen und Ergebnisse
- Die **Erklärungskomponente** (explanation component)
  kann dem Anwender begründen, durch welche Regeln und Fakten ein Ergebnis zustande kam. Sie gibt dem Experten die Möglichkeit zu überprüfen, ob das System seine Schlußfolgerungen korrekt nachbildet
- Der **Dialogteil** (dialog management)
  führt das Gespräch zwischen Anwender und Rechner
- Die **Wissensadministration**
  ermöglicht dem System zu lernen, d.h. neues Wissen in die Wissensbasis einzufügen oder altes Wissen zu verändern, ohne daß dies explizit programmiert werden muß.

Was unterscheidet ein Expertensytem von herkömmlichen Datenbanken?
Aus den bisher vorliegenden Angaben könnte abgeleitet werden, es handle sich bei einem Experten-System um nicht viel mehr als eine Datenbank mit einem komfortablen Abfragesystem. Ein Expertensystem ist aber mehr. Drei Eigenschaften, die ein Datenbanksystem nicht besitzt, charakterisieren ein Expertensystem. Es ist heuristisch, lernfähig und selbsterklärend .
Datenbanken enthalten Fakten über die reale Anwendungswelt. Nur eine kleine Anzahl von Regeln kann in Datenbanksystemen, z.B. in der Form von Integritätsbedingungen, enthalten sein. Generell ist keine Speicherung von Regeln vorgesehen. So kann bspw. die Regel „Wenn ein Student Informatik studiert, dann mag er Prolog“ in konventioneller Datenbanktechnik nicht explizit gespeichert werden. Wissensbasen erlauben im Gegensatz zu Datenbanken die explizite Darstellung regelbasierter Informationen, aus denen Schlußfolgerungen (Ableitung von Informationen) gezogen werden können.

Verfahren zur Repräsentation von Wissen können eingeteilt werden in:

1. Logische Vefahren
   Die Wissensbasis wird durch Ausdrücke der formalen Logik präsentiert. Inferenzregeln wenden dieses Wissen auf die Lösung spezifischer Probleme an. Das am häufigsten verwendete Darstellungsschema ist das **Prädikatenkalkül** 1. Ordnung. Die Programmiersprache **Prolog** stützt sich auf dieses Kalkül und ist daher für die Implementierung von Wissensbasen mit logischen Repräsentationsverfahren besonders geeignet.
2. Netzwerk-Verfahren
   Sie präsentieren das Wissen durch einen Graphen, bei dem die Knoten die Objekte und Konzepte des Problemgebiets darstellen, und die Kanten die Beziehung zwischen diesen Objekten oder Objekttypen. **Semantische Netze** sind dafür ein Beispiel. Im Gegensatz zu Datenbankmodellen (Trennung von Schema und Instanz) gehören Objekte (Instanzen) zur Repräsentation von Wissen hinzu. Netzwerkbezogene Verfahren sind demnach eine objektbezogene, graphische Darstellung von Wissen.

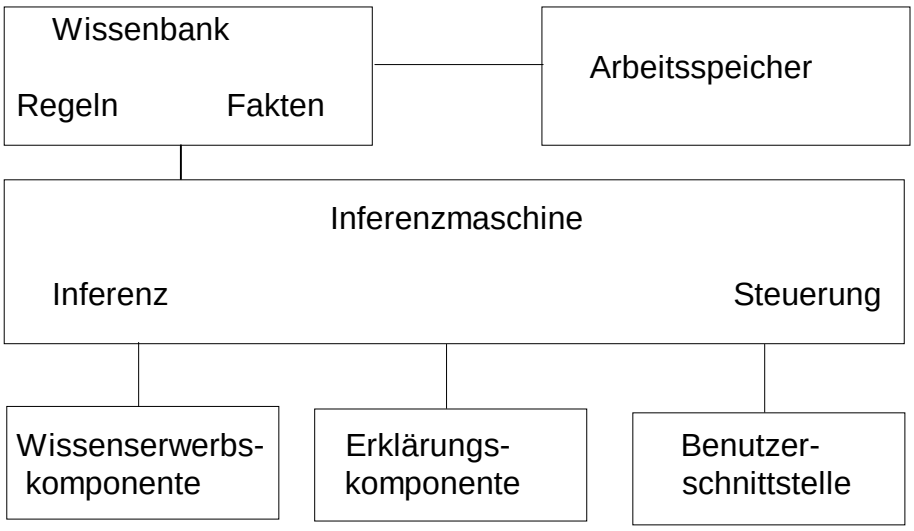


Abb. 1.2-2: Aufbau eines Expertensystems

Datenbanksysteme werden bzw. wurden für Anwendungen entwickelt, die sich folgendermaßen charakterisieren lassen:

- Die Daten weisen eine feste, vordefinierte Struktur auf
- Jeder Datensatz (jedes Tupel) beschreibt i.a. ein bestimmtes Objekt der Anwen-dung (ein Konto, ein Lagerbestand)
- Die Menge der über ein Objekt gespeicherten Daten ist i.a. klein (wenige 100 Bytes)
- Daten können mit hoher Frequenz eingefügt, modifiziert und gelöscht werden
- Es sind i.a. viele Benutzer gleichzeitig auf derselben Datenbank aktiv
- Die Daten müsen gegen Bedienungsfehler, Systemausfälle, Datenträgerverluste u.a. wirksam geschützt werden und automatisch wiederherstellbar sein

Expertensysteme wurden bzw. werden für Anwendungen konzipiert, die sich etwa durch folgende Punkte beschreiben lassen:

- Über einen bestimmten Wirklichkeitsausschnitt liegen eine Reihe feststehender Fakten und Regeln vor, sowie Gesetzmässigkeiten, Konventionen, Erfahrungen
- Das System soll dem Benutzer bei allgemeinen Problemlösungsaufgaben in der Anwendungsumgebung unterstützen, d.h.: Überprüfen von Entscheidungs-prozessen; Nachweis der Gültigkeit von Fakten; Untersuchen der Korrektheit von Aussagen
- Die hierzu benötigten Operationen sind: Auffinden einschlägiger Regeln für eine zutreffende Entscheidung; Ableitung neuer Fakten und Regeln; Bewertung von Entscheidungsvarianten
- Dem Benutzer muß eine Schnittstelle angeboten werden, an der in möglichst einheitlicher Weise die Struktur der momentanen Fakten und Regeln dargestellt und verändert werden kann.

Neben diesen anwendungsspezifischen Eigenschaften machen die meisten der heute verfügbaren Expertensysteme noch folgende Annahmen:

- Die Menge der zu verwaltenden Fakten und Regeln ist klein (klein genug, um vollständig im Hauptspeicher gehalten zu werden)
- Es arbeitet jeweils nur ein Benutzer auf derselben Fakten- und Regelmenge
- Die Sicherung und Wiederherstellung der Daten wird durch die Betriebsumgebung geleistet
- Änderung der Fakten und Regeln sind sehr selten

1.2.3 Klassifikation von Datenbanken

1.2.3.1 Klassifikationsmerkmale

Es gibt in der Datenverarbeitung zwei Arten von Informationen, die wegen ihrer grundsätzlich verschiedenen Struktur auch bei Datenbanken unterschiedliche Behandlung erfordern:

1. Texte

Jede Zeichenkette (Wort) besitzt eine Bedeutung. Die Stellung der Worte im Text ergibt den logischen Zusammenhang

2. Daten

Die Position der Daten innerhalb einer Zeichenkette weist dem Wert eine Bedeutung zu. Dadurch ist die Reihenfoge streng festgelegt.

Bsp.: Zahlenfolge '345670076'
Die Zahlenfolge sagt nichts aus, solange unbekannt ist:

- die ersten 2 Stellen (34) beinhalten den Schlüssel für eine Automarke
- die nächste Stelle (5) den Fabrikationsort
- die nächsten 4 Stellen (6700) den Autotyp

1.2.3.2 Formatierte und unformatierte Datenbanken

Formatierte Datenbestände

Die Abspeicherung der Daten erfolgt nach einem festen Schema (Format). Der Zugriff zu den Daten ist über Ordnungskriterien und Feldnamen gegeben.

Bsp.:

PERS-NR	NAME	VORNAME	,,,,,,,,,,,,,,	.....	.....
.....	.....	.....	.....	.....	.....

Unformatierte Datenbestände

Hier können Sätze nicht durch Ordnungskriterien oder feste Stellenzuordnung der Felder identifiziert werden. Deshalb müssen wichtige beschreibende Schlagworte, sog. Deskriptoren[4] festgelegt (personell oder maschinell) werden.
Bei der Suche (Information Retrieval, IR) müssen die gespeicherten Deskriptoren mit den Suchbegriffen des Benutzers verglichen werden (über Wortähnlichkeits-

[4] vgl. 1.2.2.1

prüfungen, Begriffskombinationen, Synonyme und logische Operationen sind die möglichen Fundstellen einzukreisen).

Zusammenfassung

Den grundsätzlichen Unterschied zwischen formatierten und unformatierten Daten soll das folgende Bsp. nochmals herausstellen:

Formatiert

Feldname	Pers-Nr	Name	Vorname	Beruf	Fam.-Stand	Abt.
Daten (Feldinhalte)	4711	Maier	Hans	Bäcker	verh.	10

Unformatiert

Der Mitarbeiter Hans Maier hat die Personalnummer 4711, er ist Bäcker, arbeitet in der Abteilung 10 und ist verheiratet.

Es gibt Systeme für formatierte und Systeme für unformatierte Daten. Sie sind unter folgenden Bezeichnungen bekannt:

Systeme für formatierte Daten	Systeme für unformatierte Daten
Numerische Informationssysteme	Nichtnum. Informationssysteme
Fact Retrieval	Document Retrieval
Data Base Systems	Refernce Systems
Data Management Systems	Information Retrieval Systems
Information Managament Systems	Information Storage and Retrieval
File Managent Systems	

Dokumente sind in der Regel nicht strukturiert. Sie bestehen aus einer Menge von Worten. Eine umfassende Textverarbeitung ist daher langsam und außerdem teuer, da „Textdatenbanken“ von beträchtlichem Umfang sind. So sind auch hier strukturierte Verarbeitungsformen nötig, d.h. die Darstellung des Textes muß auf Formate zurückgeführt werden. Ist das möglich, dann können die Zugriffsformen formatierter Datenbanken angewendet werden. Gewöhnlich dient dazu eine Liste von ausgesuchten Schlüsselworten je Dokument. Die Schlüsselwörter beschreiben den Inhalt des Dokuments und ermöglichen, das Dokument von anderen Dokumenten zu unterscheiden. So ergibt die Auswahl der Schlüsselwörter eine Indexierung zu den Dokumenten. Sie kann personell oder maschinell (, d.h. automatisch mit Hilfe des Rechners) erfolgen.

1.2.3.3 Daten- und Speicherstrukturen

Datenstrukturen

1. Verkettete Systeme

Verbindungen zwischen Datensätzen werden realisiert durch:

- Aufnahme von Adreßverweisen in den jeweiligen Datensatz
- spezielle sequentielle Anordnung von Datensätzen im Speicher

Satzadressen	Pers.Nr.-	Kostenstelle	Gehalt	Kettungsfelder	
				Kostenstelle	Gehalt
1	100	10	1230.-	3+	10
2	106	20	850.-	6+	7
3	110	10	1900.-	7	8
4	111	30	1600.-	5+	9
5	117	30	1400.-	8	4
6	120	20	740	9	2+
7	121	10	870	X	1
8	124	30	2400	10	X
9	130	20	1600	20	3
10	133	30	1340	X	5

Abb. 1.2-3: Adreßverkettung

2. Invertierte Systeme

Verbindungen werden über spezielle "inverted files" (Indexe, Indextafel, Wörterbücher, Adressbücher) aufgebaut. Invertierte Systeme sind weitgehend unabhängig von Datenstruktur und Zugriffspfad, da über die „inverted files“ erst im Bedarfsfall die Beziehungen aufgebaut werden.

Satzadressen	Pers.-Nr.	Kostenstelle	Gehalt
1	100	10	1230.-
2	106	20	850.-
3	110	10	1900.-
4	111	30	1600.-
5	117	30	1400.-
6	120	20	740.-
7	121	10	870.-
8	124	30	2400.-
9	130	20	1600.-
10	133	30	1340.-

Primärindextabelle Pers.-Nr.	
Feldwert	Satzadr.
100	1
106	2
110	3
111	4
117	5
120	6
121	7
124	8
130	9
134	10

Sekundärindextabelle Kostenstelle	
Feldwert	Satzadr.
10	1, 3, 7
20	2, 6, 9
30	4, 5, 8, 10

Sekundärindextabelle Gehalt	
Feldwert	Satzadr.
740.-	6
850.-	2
870.-	7
1230.-	1
1340.-	10
1400.-	5
1600.-	4, 9
1900.-	3
2400.-	8

Abb. 1.2-4: Indizierte Datei

Speicherstrukturen

Die Informationen müssen in einer Datenbank permanent gespeichert werden. Dateien spielen daher in Datenbanken eine große Rolle. Die Ausführungen über invertierte Systeme haben gezeigt, daß diese Daten nicht in einer einfachen Satzstruktur aufgebaut sind. Sogar Informationen über die Struktur der Daten müssen in Dateien gespeichert werden. Grundlage für Datenstrukturen sind Blöcke (kleinste physikalische Bearbeitungseinheit für fast alle Datenbank-Komponenten.). Die Größe eines Datenbankblocks ist nicht generell festgelegt. Sie kann beim Anlegen der Datenbank den Gegebenheiten (Anforderungen, Anwendung und Betriebssystem) angepaßt werden (gängige Werte: 2, 4 und 8Kbytes).

Blockadresse	Satzadresse	Pers.-Nr.	Kostenstelle	Gehalt
1	1	100	...	...
	2	106	...	...
	3	110	...	...
	4	111	...	...
2	5	117	...	...
	6	120	...	...
	7	121	...	...
	8	124	...	...
3	9	130	...	...
	10	133	...	...
	11	.....	...	...
	12	.....	...	...

Ist die DB einmal angelegt, dann ist die Blockgrösse eine nicht mehr änderbare Größe für die Datenbank. Für Dateien, die Daten der Datenbank enthalten, gilt

demnach: Ihre Größe entspricht immer einem Vielfachen der Datenbank-Blockgröße.
Datenbank-Blöcke stellen die kleinste Verwaltungseinheit für den Speicher dar. Ein Datenbank-Block besteht immer aus einem Kopf und dem Inhalt (z.B. den Daten-sätzen einer Datei bzw. Tabelle).

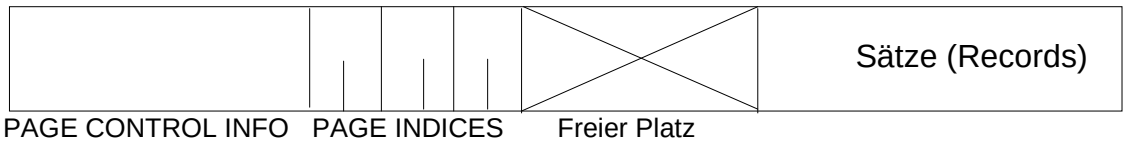


Abb. 1.2-5: Aufbau eines Datenbank-Blocks

Die logischen Strukturen der Datenbank (z.B. Tabellen und Indexe) werden in Dateien als (Datenbank-) Segment abgebildet. Jedes einzelne Segment besteht aus einer Anzahl von Blöcken (Seiten, Pages). In jedem Bereich sind die Blöcke fortlaufend numeriert, auch die Bereiche sind bekannt. Alle Datensätze der Datenbank können ermittelt werden, falls die Blocknummer des Blocks, in dem die Daten gespeichert sind, und das Segment, in dem sich der Block befindet, bekannt sind (physikalischer Databasekey).
Segment-Typen werden in drei Gruppen unterteilt:

1. Segment-Typen, die intern benötigt werden
2. Segment-Typen, die Benutzer-Daten (Dateien, Tabellen) aufnehmen
3. B*-Indexe, die zum schnellen Auffinden der Daten oder Sortierfunktionen genutzt werden.

Zur Speicherung und Verwaltung der Indexe werden B*-Baüme verwandt:

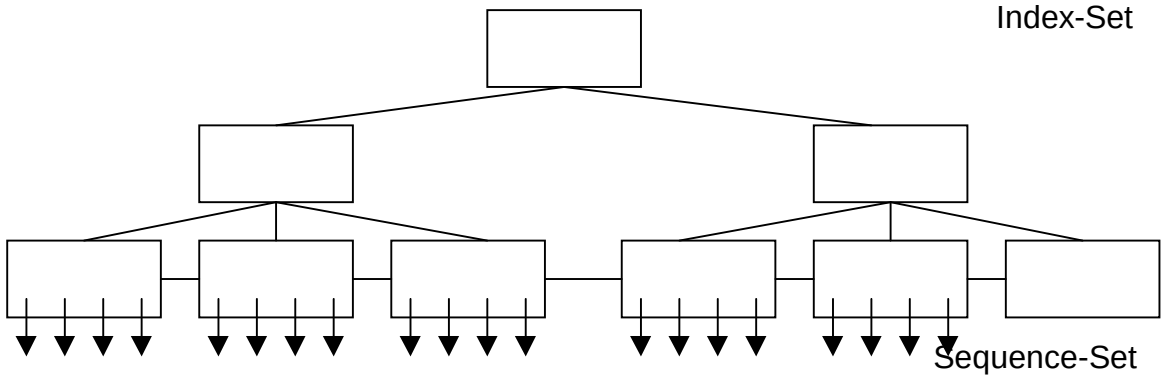


Abb. 1.2-6: Aufbau eines B*Baums

Die Knoten eines B*-Baums lassen sich in den sog. `Sequence Set` (Bereich der Bätter), der Schlüssel mit den Adressen der zugehörigen Datenblöcke in sortierter Reihenfolge enthält und den `Index Set` aufteilen, der den schnellen, gezielten Zugriff auf die Blöcke des `Sequence Set` ermöglichen soll. Ein typischer Zugriff auf einen B*-Baum beginnt bei der Wurzel und arbeitet sich über die Zeiger des Index Set an die betreffenden Knoten des `Sequence Set` heran. Dort werden die dem gesuchten Schlüsselwert entsprechenden Satzadressen gelesen (Databasekeys) und der Zugriff auf die Blöcke der zugeordneten Tabellen (Dateien) durchgeführt.

1.2.3.4 Grundfunktionen der Datenbanksoftware

Betriebssysteme bieten nur Zugriffsmethoden zu einfachen Dateien an. Zur Datenbank gehört demnach Verwaltungssoftware mit den Aufgaben:

- Verwaltung der Speicherstruktur
- Entfernen bzw. Einordnen der gewünschten Daten.

Man spricht von einem Datenbankverwaltungs-Programm oder **Data Base Management System** (**DBMS**).
Die Nutzung eines DBMS kann auf 2 Arten erfolgen:

1. Die Nutzung des DBMS findet zwischen Endbenutzer und dem DBMS auf direktem Wege statt. Es handelt sich dabei um ein geschlossenes System aus Datenbank und Daten-bankverwaltung. Die Systeme werden self-contained, exekutiv oder anwen-dungsorientiert genannt.
2. Die Kommunikation mit der Datenbank erfolgt über Programme.
   Die Bedienung eines solchen eingebetteten oder "host-language"-Systems erfordert Programmierkenntnisse, denn Datenbankanweisungen und Datenbank-beschreibung richten sich nach den Konventionen der Gastsprache der Benutzerprogramme. Man spricht von programmorientierten, operierenden bzw. "host-language" - Systemen.

Weiterhin muß

- es eine Methode zur Strukturdefinition der Datenbank (Design) geben.
- das Design in einer Struktur-Definitions-Sprache (**Data Description Language**, **DDL**) als Quellcode-Schema deklariert werden. Es muß ein DDL-Übersetzer (DDL-Prozessor) zur Verfügung stehen, um Objektcode zu erzeugen.
- es Ladebefehle (Laden der Datenbank) bzw. allg. "Utilities" zurAdministration geben
- eine Redefinition der Struktur leicht möglich sein
- zur Ein-/Ausgabe von Daten der Datenbank eine Möglichkeit zur Daten-manipulation vorhanden sein. Diese Möglichkeit kann in einer **Datenmanipulations-Sprache** (**DML**) vorliegen, die in eine gastgebende, herkömmliche Programmiersprache (Cobol, Pascal, C) eingebettet ist. Es kann aber auch eine eigenständige (meistens abfrageorientierte) Sprache (QL) vorliegen. In vielen Datenbanksystemen gibt es gleichzeitig beide Möglichkeiten.

Zu diesen Datenbankaufgaben kommt noch als weitere Aufgabe hinzu: System-steuerung zur Koordinierung der verschiedenen Aufträge innerhalb des Datenbanksystems und Datenkommunikation zum Nachrichtenaustausch mit Außenstellen (**Transaktionsbetrieb**).

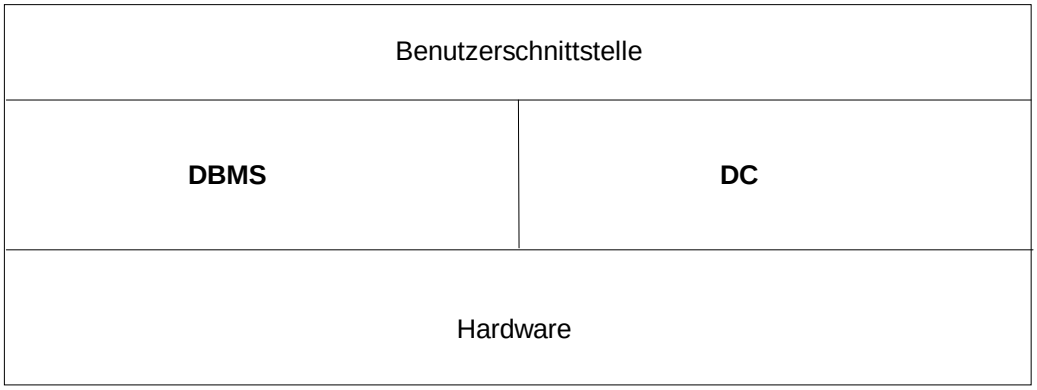


Abb. 1.2-7: Überblick zu den Komponenten eines DB/DC-Systems

Aufgaben eines DC-Systems

- Bereitstellen einer Schnittstelle für Anwender-/Datenbankprogramm, die die Eigenheiten der Datenstationen und die techn. Eigenschaften der Datenübertragung verdeckt.
- Abwicklung des Nachrichtentausches zwischen einer Vielzahl von Datenstationen und den Anwender-/Datenbankprogrammen
- Weiterleitung der Nachrichten in Abhängigkeit von Priorität, Betriebsmittel-verfügbarkeit und Schutzregelungen an den Empfänger

Der **Fernzugriff auf Datenbanken** ist im wesentlichen durch 4 Komponenten bestimmt:

- den Bildschirmarbeitsplatz
  Das kann bspw. ein Bankautomat mit einfacher Benutzeroberfläche oder ein komplexes System (CAD-Workstation mit leistungsfähigem Prozessor) sein
- das Anwendungsprogramm
  Das ist in der Regel ein transaktionsorientiertes Programm[5], das einfache Abfragen und Änderungswünsche entgegennimmt, Zugriffsberechtigung des Benutzers prüft, Datenbankzugriffe abwickelt und dem Benutzer das Resultat übermittelt.
- das Datenbanksystem (DB-System, DBS)
  Es besteht in der Regel aus mehreren Prozessen. Diese Prozesse erhalten ihre Aufträge von Anwendungsprogrammen (, die auch als eigene Prozesse ablaufen). Man unterscheidet Prozesse, die spezielle Aufgaben ohne Zuordnung zu einezelnen Anwendern ausführen (Hintergrundprozezesse) von Prozessen, die direkt (1:1) den angemeldeten Benutzern zugeordnet sind (Schattenprozesse, "Dedicated Server-Prozesse). All diese Prozesse sind Prozesse des Server (Backend). Die Versorgung des Backend mit Aufgaben übernimmt mindestens ein Frontend-Prozeß. Frontend-Prozesse werden auch als Client oder Anwendung bezeichnet.

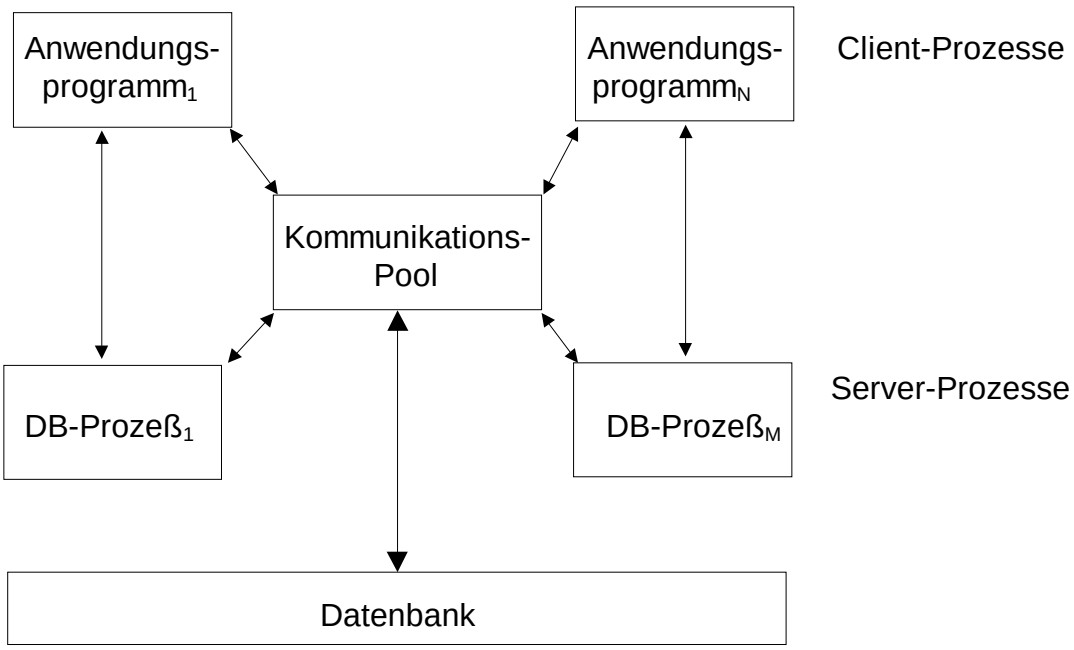


Abb. 1.2-8: Interprozeß-Kommunikation Anwender- bzw. Datenbank-Prozesse

Zur Abarbeitung gleichzeitiger Aktionen dient eine Datenverarbeitungszentrale im Hauptspeicher (Communication Pool, System Clobal area). Alle Daten, die das Datenbanksystem für mehrere verschiedene Aufgaben benötigen könnte, werden hier zwischengespeichert.

[5] vgl. 1.5.2

DB-Prozeße bearbeiten Transaktionen. Unter **Transaktion** versteht man eine Folge von Datenbankzugriffen, die eine Datenbank von einem gültigen Zustand in einen neuen, ebenfalls gültigen Zustand überführen. Die Aktionen einer Transaktion sind aus Sicht des DBS die kleinsten ausführbaren (atomaren) Einheiten, deren korrekte Ausführung das DBS übernimmt.

- das Kommunikationssystem (DC-System)
  Es übernimmt die Verbindung der einzelnen Komponenten. Drei Fälle können unterschieden werden:

1. Terminalnetze
   Beim Anschluß "dummer Terminals" über größere Entfernungen dient das Netz als "Verlängerungsschnur". Auf der Seite des Terminals müssen Kommunikations-protokolle bereitgestellt werden, die wegen ihres geringfügigen Aufwands in Hardware implementiert werden können. Auf der Seite des Rechners liegt der Schwerpunkt der Verarbeitung (z.B. Aufbau und Kontrolle der Verbindung, Program-mierung des Bildschirms).
2. Netze zwischen autonomen Rechnern
   Zum Anschluß von "Intelligenten Bildschirmarbeitsplätzen" (z.B. PC, CAD-Arbeitsstation) mit integriertem Rechner muß ein Kommunikationsnetz zwischen eigenständigen Rechensystemen bereitgestellt werden.
3. Verteilte Datenbanken
   Das Netz dient zur Kommunikation der einzelnen Datenbankverwal-tungssysteme.

1.2.3.5 Data Dictionary / Directory System (DD/D-System)

Es enthält Informationen über Definition, Struktur und Benutzung von Daten. Das "`directory`" liefert dem eingesetzten DB-System Informationen über Plazierung und Strukturen innerhalb der Benutzerdatenbank (DB-System orientiert). Ein „**data dictionary**“ dagegen gibt dem Anwender Information über Daten, ihre Herkunft, wo sie genutzt werden und wann sie geändert wurden (zentralisierte Ablage von Informationen über Datendefinitionen).
Die meisten in einer Installation benötigten Daten befinden sich irgendwo in Bibliotheken, Dateien, Verzeichnissen oder Listen. In der Regel sind diese Informationen jedoch nicht hinreichend miteinander verbunden. Ein „**data dictionary** (**DD**)“ ist ein Hilfsmittel, bei dem alle Definitionen in (einer Gruppe von) Datenbanken gespeichert werden, um leichter Fragen beantworten zu können, Auswertungen durchführen und Ausgaben erzeugen zu können.

Ein DD enthält:

- Angaben über Definition, Herkunft, aktuelle Benutzung und Änderung, Struktur und Benutzungsvorschriften der Daten, z.B.: Namen, zulässige Wertebereiche, logische Beziehungen, Integritäts- und Zugriffsmöglichkeiten, Namen und Eigenschaften von Anwenderprogrammen mit Angaben über Speicherung, Codierung und Auffinden der Daten (Adreß und Längenangaben, Feldtypen, Zugriffspfade und physische Plazierung in der Datenbank)
- Angaben über Speicherung, Codierung und Längenangaben, Feldtypen, Zugriffspfade in der Datenbank.
- Angaben zu Verarbeitungseinheiten (Modul, Programm, Segment)

Funktionen eines DD sind:

- Erstellen von Berichten (z.B. Benutzerreports: welche Daten werden von wem benutzt?)
- Generierung von Datendefinitionen und Datenbank-Beschreibungen (Datenmodellierung) aus Definitionen im **DD**

Vorteile des **DD** bestehen auf vier Gebieten: Dokumentation, Datenbankdefinition, Anwendungsentwicklung und Kontrolle

Dokumentation
Wird das DD richtig eingesetzt, so wird es bspw. die Definition des Felds Pers.-Nr. nur einmal geben und diese Definition befindet sich im **DD**. Im **DD** werden De-finitionen nicht mehrfach gespeichert, sondern entsprechend untereinander ver-knüpft.

Datenbankdefinition
Definition zu Datenbanken werden zweckmäßig im **DD** angelegt und verwaltet.

Anwendungsentwicklung
Das **DD** ist eine Hilfe festzustellen, ob bestimmte Datendefinitionen bereits im **DD** vorhanden sind oder nicht. Außerdem besteht die Möglichkeit Satzstrukturen im neutralen Format des **DD** anzulegen.
Kontrolle
Das **DD** ist eine Hilfe, Definitionen eindeutig zu machen.

Dienstprogramme zur Auswertung eines **DD** ermöglichen

- die Analyse der vorhandenen Datenobjekte (z.B.: die Bestimmung von Redun-danzen, Inkonsistenzen)
- "Cross-Reference- " Listen
- Statistiken
  (Benutzungshäufigkeiten, Werteverteilungen) zur Optimierung der Speicher-struktur

Ein **DD** läßt sich, falls alle Daten über Dateien und Programmsysteme in das Wörterbuch aufgenommen werden, zur zentralen Kontrollinstanz über alle gespeicherten Daten ausbauen.

Man spricht häufig auch von aktiven und passiven „Data Dictionaries“ und bezieht sich dabei auf den Grad der Integration des Data Dictionary mit dem Datenbank-verwaltungssystem (DBMS). Falls das DBMS die Definition des **DD** zur Laufzeit von Programmen auf den neuesten Stand bringt, liegt ein aktives **DD** vor. Ist dazu ein eigenständiger Regenerationslauf nötig, handelt es sich um ein passives **DD**.
In beiden Fällen ist dann die Architektur einer Datenbank[6] um das Datenwörterbuch (**DD**) organisiert:

[6] vgl. 1.2.4.6

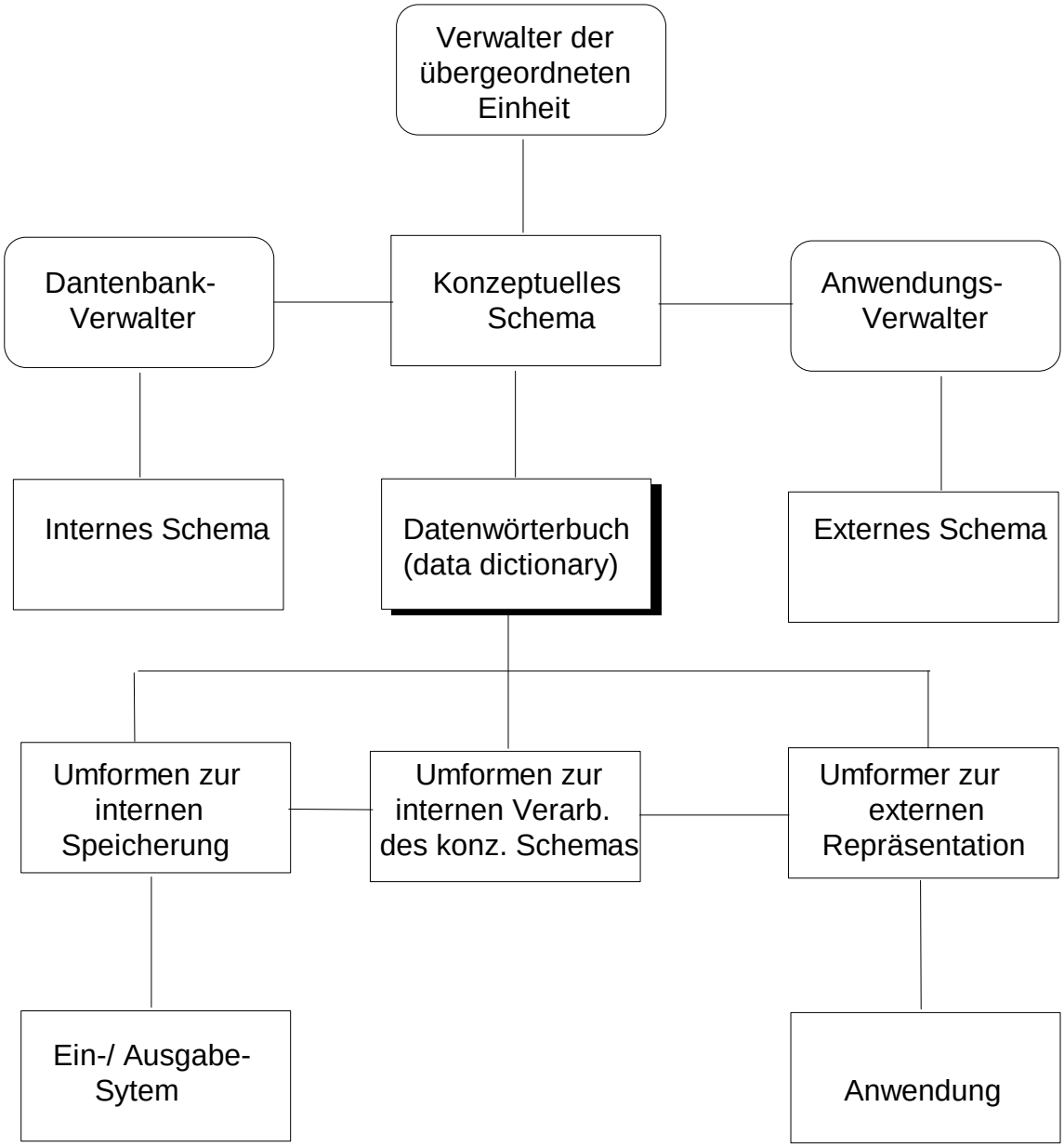


Abb.: 1.2-9: Organisation einer DB-Architektur um das DD

1.3 Datenbankmodelle für formatierte Datenbanken (DB)

Sie werden danach eingeteilt, wie die elementaren logischen Einheiten der Datenbank, die Datensätze, zu Datenstrukturen (Baum, einfaches Netz, Tabelle) zusammengefaßt sind.

1.3.1 Beschreibung der Daten in formatierten Datenbanken

1.3.1.1 Entitätsmengen und ihre Beziehungen

1. **Grundlagen**

Durch eine Datenbankanwendung sind die unterschiedlichen Aufgaben eines Bereichs zu koordinieren. Ein derartiger Bereich ist eine umfassende Verwaltungseinheit, z.B.:

- ein Industrieunternehmen (mit Produktionsdaten)
- eine Bank (mit Daten diverser Konten)
- ein Krankenhaus (mit Daten über Patienten)
- eine Hochschule (mit Daten über Studenten/ Dozenten)

Bsp: „Produktionsdaten eines Unternehmens“
Informationen werden gewünscht über

- die gegenwärtig vorhandenen Projekte
- die Bauteile/Teile, die zur Durchführung dieser Projekte benötigt werden
- die Lieferanten, die die Bauteile bereitstellen
- die Angestellten, die an diesen Projekten mitarbeiten

Projekte, Lieferanten, Bauteile, Angestellte sind die elementaren Einheiten, über die Daten in der Datenbank gespeichert sind.
Das ergibt folgenden Schemaentwurf:

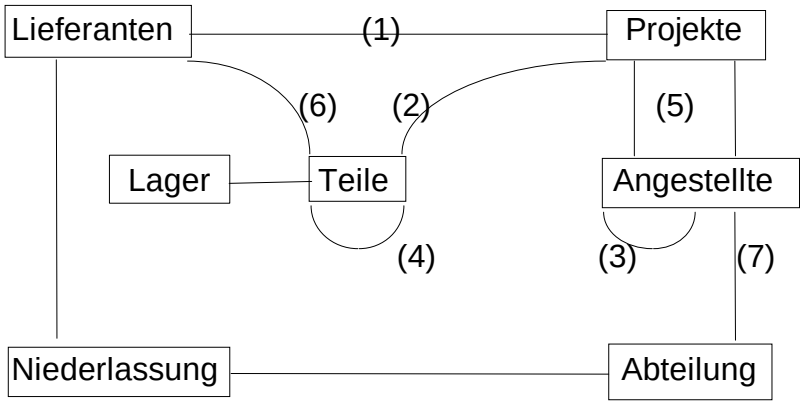


Abb. 1.3-1: Schema für die Produktionsdaten eines Unternehmens

Die Beschreibung eines Bereichs der realen Welt ist nur durch Abstraktion vieler konkreter Gegebenheiten und geeignete Modellbildung möglich.

Modellbildung in einem Datenbanksystem bedeutet:

Gewisse Dinge (**Objekte**) der realen Welt werden herausgestellt, da sie zur Problemlösung benötigt werden. Zwischen den Objekten existieren **Beziehungen**. Anstatt des Begiffs Objekt benutzt man hier auch häufig den Begriff **Entität** (entity).[7]

2. **Was ist eine Entität**?

Eine **Entität** ist ein individuelles oder identifizierbares Exemplar von Dingen, Personen oder Begriffen der realen Welt.
Eine **Entität** kann demnach sein:

- ein Individuum (z.B. ein Student, ein Dozent, ein Mitarbeiter, ein Einwohner)
- ein reales Objekt (z.B. eine Maschine, ein Gebäude, ein Produkt)
- ein abstraktes Konzept (z.B. eine Fähigkeit, eine Vorlesung)
- eine Beziehung

Wird ein Individuum, ein reales Objekt, ein abstraktes Konzept, ein Ereignis oder eine Beziehung als Entität deklariert, so wird damit ausgedrückt: „Sachverhalte, die den als Entität deklarierten Begriff betreffen, sollen speicherbar sein, sobald ein eindeutig **identifizierendes Merkmal** bekannt ist."
Entitäten können in verschiedenen **Entitätstypen** klassifiziert werden, z.B. Studenten, Vorlesungen, Angestellte. Eine Entität wird durch ihre **Attribute** beschrieben. Einer Entität werden also Merkmale (Eigenschaften) zugeordnet, die zu seiner Beschreibung wichtig sind.

Bsp.: „Die Produktionsdaten eines Unternehmens"
Zur Beschreibung des Entitätstyp Angestellte ist notwendig: NAME, BERUFSBEZEICHNUNG
Zur Beschreibung des Entitätstyp TEILE kann herangezogen werden: BEZEICHNUNG, FARBE, KOSTEN, GEWICHT.
BEZEICHNUNG, FARBE, GEWICHT sind Eigenschaftsnamen (Attribut-Namen).

Jedes **Attribut** kann Werte aus einem bestimmten Wertebereich (**Domäne**) annehmen. Eine **Domäne** stellt die Menge aller möglichen Werte für eine spezifische Eigenschaft bereit, z.B.

NAME	ALTER	GEWICHT	BEURTEILUNGSKRITERIUM	STUDIENFACH
Karl	20	66	gut	Mathematik
Fritz	25	75	mittel	Chemie
Juergen	27	68	schlecht	Informatik
Josef	24	87		Physik

Darstellung von Entitäten

[7] vgl. Vetter, M.: Aufbau betrieblicher Informationssysteme, 6. Auflage Stuttgart 1990

Jedem Entitäts-Typ kann man eine Kombination von Attributen, jeder Entität dieses Typs eine entsprechende Kombination von Attributwerten zuordnen.
Bsp.: „Produktionsdaten eines Unternehmens“

Dem Entitätstyp ANGESTELLTE kann zugeordnet werden:

```
NAME: Karl
BERUFSBEZEICHNUNG: Programmierer
NAME_DER_KINDER: Fritz, Anna
```

In der Regel sind Attributkombinationen so beschaffen, daß sie die zugehörige Entität eindeutig beschreiben.
Alle Entitäten des gleichen Typs bilden eine Entitätsmenge
Bsp.: "Alle Studenten einer Hochschule" (charakterisiert durch die Eigenschaften NAME, ALTER, WOHNORT)

Ein **Entitäts-Schlüssel** (Schlüsselkandidat, „`candidate key`“) ist dann ein Attribut, das Entitäten innerhalb einer Entitätsmenge eindeutig identifiziert. Ein Entitätstyp kann mehrere Schlüsselkandidaten haben. Es ist sinnvoll, aus den Schlüsselkandidaten einen **Primärschlüssel** („`primary key`“) auszuwählen. Soweit es möglich ist, sollte man zusammengesetzte Primärschlüssel vermeiden.

3. **Beziehungen**

Zwischen Entitäten können Beziehungen bestehen. So sind bspw. in den „Produktionsdaten eines Unternehmens“ Beziehungen:

(5) Ein Angestellter ist Mitarbeiter an bzw. ist Projektleiter von bestimmten Projekten
(7) Ein Angestellter gehört zu einer bestimmten Abteilung

Gleichartige Beziehungen können zu einer Beziehungsmenge (bzw. zu einem Beziehungstyp) zusammengefaßt werden.

4. **Klassifizierung von Beziehungen**

Es können identifizierende oder beschreibende Attribute unterschieden werden.

Ist E bzw. E' die Menge der identifizierenden Attributwerte und W die Menge der be-schreibenden Attributwerte, dann existieren folgende Abbildungsmöglichkeiten:

1) Beziehungen zwischen identifizierenden und beschreibenden Attributwerten ($E \rightarrow W$).
2) Beziehungen zwischen identifizierenden Attributwerten verschiedener Entitätsmengen ($E \rightarrow E'$)
   Bsp.: „Produktionsdaten eines Unternehmens“
   (6) Lieferanten-Nr. $\rightarrow$ Teile-Nr.
   Ein Vorkommen des Lieferanten L1 identifiziert bspw. eine bestimmte Anzahl (0 inbegriffen) von Teilen T1, T2, T3.
   Ein Vorkommen T2 identifiziert bspw. eine bestimmte Anzahl von Lieferanten (L1, L3).
3) Beziehungen zwischen identifizierenden Attributwerten der gleichen Menge ($E \rightarrow E$).

Bsp.: "Produktionsdaten eines Unternehmens"
(3) Unterstellungsverhältnis
Es zeigt, welche Angestellte Vorgesetzte bzw. Untergebene sind.
(4) Stückliste
Sie zeigt, welche Unterteile (Komponenten) montiert werden, um ein Oberteil zu erhalten
(4) Verwendungsnachweis
Er zeigt, in welche Oberteile (Baugruppen) eine Komponente eingeht.

Häufig ist die Einteilung komplexer Strukturen ($E \to E$, $E \to E'$) abhängig von der Abgrenzung der Entitätsmengen., z.B.:

a) Die Beziehung Pers.-Nr. des Ehemanns zur Pers.-Nr. der Ehefrau ist von der Art $E \to E$, wenn alle Personen einer Menge zusammengefasst werden.
b) Erfolgt die Zusammenfassung getrennt nach Männern und Frauen, so liegt die Art $E \to E'$ vor.

5. **Kardinalitätsverhältnis**

Es gibt darüber Auskunft, in welchem Verhältnis zwei Entitätstypen über einen bestimmten Beziehungstyp miteinander verbunden sind.

Man unterscheidet:

Beziehungen vom Verhältnis 1:1
Zwischen zwei Entitätstypen (Entitätstyp 1 und Entitätstyp 2) kann zu einem Zeitpunkt nur eine einzige Entität aus Entitätstyp 1 mit einer einzigen Entität aus Enti-tätstyp 2 in Verbindung stehen. So ist der Beziehungstyp "Heirat" eine 1:1 - Verbindung zwischen zwei "Personen"-Entitäten.
Beziehungen vom Verhältnis 1:N
Zwischen den beiden Entitätstypen kann zu einem Zeitpunkt nur eine einzige Enti-tät aus Entitätstyp 1 mit mehreren Entitäten aus Entitätstyp 2 in Verbindung stehen. So ist bspw. der Beziehungstyp zwischen Abteilung und Angestellte[8] vom Verhältnis 1:N.
Beziehungen vom Verhältnis M : N
Zwischen zwei Entitätstypen können zu einem Zeitpunkt mehrere Entitäten aus Entitätstyp 1 mit mehreren Entitätstypen aus Entitätstyp 2 (über den Beziehungstyp) in Verbindung stehen.
So ist bspw. der Beziehungstyp zwischen Projekte/Teile vom Verhältnis M:N.

6. **Mitgliedsklassen**

Ergibt sich aus den Anforderungen des Anwendungsgebiets, daß jedes Exemplar eines Entitätstyps an einer Beziehung teilhaben muß, dann wird diese Klasse des Entitätstyps in dieser Beziehung als "**mandatory** (zwingend)" bezeichnet. Andernfalls ist die Mitgliedsklasse "**optional** (freigestellt)".
Bsp.: Eine Datenbank für die Verwaltung einer Gemeinde umfaßt Bürger (alle Einwohner) und Haushaltsvorstände (zweckmäßigerweise sind das die Einwohner der Gemeinde von denen man Steuern erheben kann). Jeder Bürger ist zwingend mit dem Datum der Geburt Einwohner der Gemeinde

[8] vgl. Abb. 1.3-1

und bleibt das, bis er "gelöscht" wird (z.B. beim Tod oder einem Umzug). Zum Steuerzahler wird er dann (im übertragenene Sinne optional), falls er Grundbesitz (Grundsteuer) hat oder ein Gewerbe betreibt.

Die Entscheidung, ob eine Mitgliedsklasse eines Entitätstyps in einer Beziehung zwingend oder freigestellt ist, liegt häufig im Ermessen des Datenbank-Modellierers.

1.3.1.2 Beziehungen und Beziehungsattribute

Ein Beziehungsattribut beschreibt die Verknüpfung einer Beziehungsmenge mit einer Domäne bzw. mehreren Domänen, z.B.:

Beziehungen zwischen Studenten (repräsentiert durch den Entitätsschlüssel `S#` mit den Erscheinungsformen `{S1, S2, S3}` und Dozenten (repräsentiert durch den Entitätsschlüssel `D#` mit den Ausprägungen `{D1, D2}` werden durch die Beziehungs-menge `BELEHRT` beschrieben, der das Beziehungsattribut `BEURTEILUNG` zugeordnet werden kann. So kann bspw.

```
BELEHRT
   (D1, S1)
   (D1, S3)
   (D2, S1)
   (D2, S2)
   (D2, S3)
```

über das Beziehungsattribut Werte aus dem Wertebereich `KRITERIUM gut, mittel, schlecht` zugeordnet erhalten.
Das Attribut `BEURTEILUNG` verknüpft die Beziehungen `BELEHRT` mit der Domäne (Beurteilungs-) `KRITERIUM`. Ein spezifisches Dozenten-Studenten-Beziehungspaar steht mit einem einzigen spez. Kriterium in Beziehung (Assoziation einfacher Art).Beziehungen werden identifiziert, indem man die Schlüssel der Entitäten in der Beziehung nutzt.

1.3.2 Das relationale Datenbankmodell

Grundlage dieses Datenmodells sind einfache Tabellen, z.B.:

Schlüssel	Name	Vorname	Fakultät	Geburtsdatum
110506	.....	Werner	Medizin	10.03.71
060313	.....	Karl	Jura	10.01.73
2517008	.....	Fritz	Mathematik	09.03.70

Abb. 1.3-2: Schematische Darstellung zu einem relationalen DB-Modell

Die Tabelle wird auch **Relation** genannt. Datenbanken, die aus solchen Relationen aufgebaut sind, heißen **relationale Datenbanken**. Die Datenbanken bestehen aus "flachen", hierarchiefreien Zusammenstellungen von Datenfeldern. Die Spalten der Tabelle bezeichnet man als Attribute der Relation. Eine flache Datei (`flat file`) ist eine Relation und besteht aus einer Menge von Tupeln (Tabellen-Zeilen). Zu einem Tupel sind die auf ein bestimmtes Objekt bezogenen Datenwerte zusammengefaßt.
Das relationale Datenbankmodell[9] besitzt im wesentlichen folgende Eigenschaften[10]:

1) Die Daten werden einheitlich durch Werte repräsentiert, die in Form von Tabellen dargestellt werden.
2) Der Benutzer sieht keine speziellen Verweisstrukturen zwischen den Tabellen.
3) Es gibt Operationen zur Auswahl von Tabellenzeilen (Selektion), Tabellenspalten (Projektion) sowie zur Verbindung ("`join`") von Tabelleneinträgen. Keine dieser Operatoren ist jedoch auf Kontrollstrukturen angewiesen oder durch vordefinierte Zugriffsstrukturen beschränkt. Diese Operationen werden über eine Datenmanipulationssprache (**DML**[11]) realisiert, die auf Daten zugreifen, Daten einfügen, löschen, korrigieren und Anfragen beantworten kann. Datenmanipulationssprachen im Bereich relationaler Datenbanken sind besonders leicht und einfach. Ein weit verbreitetes Beispiel dafür ist SQL[12].

Die Darstellung der Daten in einer relationalen Datenbank folgt speziellen Vorschriften. Diese Vorschriften (Theorie des relationalen Datenbankentwurfs) bezeichnet man üblicherweise als **Normalisierungstheorie**[13]. Normalisieren bedeutet: Darstellung des logischen Schemas einer relationalen Datenbank in der Form einfacher, nicht geschachtelter Tabellen.
Das relationale Datenbankmodell ist heute die übliche Organisationsform, in der Daten für die Abbildung im Rechner beschrieben werden. Für die Darstellung gelten allgemein folgende Regeln:

- Alle Datensätze (Tabellenzeilen) sind gleich lang
- Jeder Datensatz (Tupel) kommt nur ein einziges Mal in einer Tabelle vor, die Reihenfolge der Sätze (Tupel) ist beliebig. Jede Tabellenzeile beschreibt einen Datensatz. Die eindeutige Identifizierung eines Tupels erfolgt über den „Schlüssel“
- Der Wert eines Attributs kommt aus einem bestimmten Wertebereich (Domäne). Jede Tabellenspalte beschreibt eine bestimmte Eigenschaft (Attribut) des Datenobjekts.
- Durch Kombination einzelner Spalten (Attribute) und Zeilen (Datensätze) können neue Tabellen (Relationen) gebildet werden.

Im wesentlichen heißt **Normalisieren** Beseitigung der funktionalen Abhängigkeiten, die zwischen den einzelnen Attributen eines Relationenschemas vorliegen können. Eine **funktionale Abhängigkeit** ist zwischen Attributmengen, z.B. A_i und A_j dann gegeben, falls in jedem Tupel der Relation der Attributwert unter A_i -Komponeneten den Attributwert unter den A_j-Komponenten festlegt. Die funktionale Abhängigkeit wird dann so beschrieben: $A_i \rightarrow A_j$.

---

[9] vgl. Codd, E.F.: "A Relational Modell of Data for Large Shared Data Banks", Communications of the ACM, June 1970, Seiten 377 - 387

[10] Datenbanksysteme, die auf diese Konzepte beschränkt sind, werden als "minimal relational" bezeichnet.

[11] vgl. 1.2.3.4

[12] vgl. 1.4.3.2

[13] vgl. 2.1.1

Schlüssel sind Spezialfälle funktionaler Abhängigkeiten. Ein Schlüssel X beschreibt für ein Relationenschema eine funktionale Abhängigkeit, wenn X minimal ist (d.h. aus der kleinsten Menge Tupel identifizierender Attributwerte gebildet ist). Ziel eines sich auf Abhängigkeiten abstützenden Datenbankentwurfs einer relationalen Datenbank ist: Umformen aller funktionalen Abhängigkeiten in Schlüssel-abhängigkeiten. Die Menge der Abhängigkeiten ist äquivalent zur Menge der Schlüsselbedingungen im resultierenden Datenbankschema (**Abhängigkeitstreue**).
Bsp.: Gegeben sind in der folgenden Tabelle die Studiendaten von Studenten an einer Hochschule (, Schlüssel sind MATNR und SRNR).

STUDIENDATEN[14]

MATNR	NAME	ADRESSE	SRNR	STUDR.	ANFANGSDAT
123	Meier	Berggasse 19	88	Informatik	01.10.1989
124	Müller	Lange Gasse 19	81	Physik	01.03.1989
123	Meier	Berggasse 19	79	Psychologie	01.10.1985
125	Schmidt	Grabenweg 4	79	Psychologie	01.03.1990
128	Lang	Brückenweg 23	88	Informatik	01.10.1989
129	Lang	Brückenweg 23	81	Physik	01.10.1989
127	Bauer	Hochstraße	88	Informatik	01.10.1977

Abb. 1.3-3: Tabelle mit Studiendaten

Die vorliegende Tabelle zeigt redundante Daten (Informatik, Physik, Psychologie). Die Abhängigkeiten dieser Daten (Bezeichnung der Studienrichtung) vom Attribut Studienrichtungsnummer ist offensichtlich, zweckmäßigerweise ist diese Abhängigkeit in einer eigenen Tabelle zu verwalten. Die Attribute NAME, ADRESSE sind vom Teilschlüssel MATNR abhängig, die Attribute STUDR, ANFANGSDATUM vom Teilschlüssel Studienrichtungsnummer abhängig. Es ist besser, solche Abhängigkeiten in getrennten Tabellen zu verwalten.
So führt die Normalisierung der vorstehende Tabelle auf folgende Tabellen:

STUDENT

MATNR	NAME	ADRESSE
123	Meier	Berggasse 19
124	Müller	Lange Gasse 19
125	Schmidt	Grabenweg 4
127	Bauer	Hochstraße 2
128	Lang	Brückenweg 23
129	Lang	Brückenweg 23

STUDIUM

MATNR	SRNR	ANFANGSDAT
123	88	01.10.1989
124	81	01.03.1989
123	79	01.10.1985
125	79	01.03.1989
127	88	01.10.1977
128	81	01.10.1989
129	88	01.10.1989

[14] Abkürzungen: MATNR...Matrikelnummer SRNR...Studienrichtungsnummer
STUDR...Studienrichtung ANFANGSDAT...Anfangsdatum

STUDIENRICHTUNG

SRNR	STUDR
88	Informatik
81	Physik
79	Psychologie

Abb. 1.3-5: Normalform der Tabelle mit Studiendaten

Der vorliegende Zerlegungsprozeß zeigt eine geeignete Auswahl von Spalten der Ausgangstabelle. Normalisieren heißt demnach auch: „Zerlegen von Relationenschemata in kleinere, übersichtlichere Tabellen“. Es sind nur sinnvolle Zerlegungen (Kriterium: funktionale Abhängigkeiten) zugelassen, d.h.: Die Ausgangsrelation muß sich (ohne Informationsverlust) durch Zusammenfügen von Teilrelationen der Zerlegung rekonstruieren lassen (**Verbundtreue**). Zerlegungen sind nur dann sinnvoll, wenn sie verlustfrei sind. Beim Zusammensetzen der Teilrelationen müssen wieder genau die Tupel der Ausgangsrelation erzeugt werden. Es sollen aber möglichst wenige Tabellen erzeugt werden, die den Forderungen genügen (**Minimalität**).
Eine Menge funktionaler Abhängigkeiten beschreibt die Integritätsbedingungen über dem Relationenschema. Sie wird bei der Zerlegung des Relationenschemas mit folgender Einschränkung auf die Schemata der Tabellen vererbt: Über einem Teilschema dürfen nur Abhängigkeiten definiert sein, deren Attribute vollständig im Teilschema enthalten sind. Zum anderen sollte die Menge der Abhängigkeiten, die über dem Schema gültig sind, zusammengenommen äquivalent zu der Menge über dem Ausgangsschema sein (**abhängigkeitsbewahrende Zerlegung**).
Ein anderer Vorteil der Zerlegung größerer Tabellen in kleinere Tabellen ist die leichtere Definition von Sichten (Views). Ein „**View**“ ist eine Relation, die nicht explizit in der Datenbank gespeichert ist, sondern aus den gespeicherte Tabellen abgeleitet wird. Nachdem ein „View“ definiert wurde, kann er in gleicher Weise benutzt werden wie jede andere Relation.
Die Suche in einer relationalen Datenbank erfolgt durch Angabe von bestimmten Attributwerten (eingebettet in Abfragesprachen). Da das Durchsuchen aller Daten-sätze viel zu lang dauern würde, muß die für das Suchen wichtige Information über einen Index herausgezogen werden. Im Index steht neben der Suchinformation dann ein Verweis auf den vollständigen Datensatz. Selbstverständlich ist der Index so einzurichten, daß die Suchzeit möglichst kurz ist. Außerdem belegt der Index zusätzlichen Speicherplatz. Man kann natürlich auch mehrere Indexe für verschiedene Suchkriterien einrichten. Die Architektur einer relationalen Datenbank ist somit intern wesentlich durch die Organisation der Daten- und Indexdatei(en) bestimmt.
Durch Integration der logischen Programmierung (Prolog) mit relationalen Datenbanken entstehen **deduktive Datenbanken**. Sie ergänzen die Fakten der relationalen Datenbank (Tupel) mit Regeln. Relationale Datenbanken werden dadurch zu Wissensbasen. Die zur Manipulation logischer Datenbanken entwickelten Datenbanksprachen (z.B. Datalog) stützen sich auf die **Prädikatenlogik**.

1.3.3 Das Entity-Relationship Modell

Grundlagen

Das Entity-Relationship Modell [15] von Chen ist ein Versuch, Beziehungen zwischen den Daten zu formalisieren. Die Objekte der Wirklichkeit, die durch die Daten modelliert werden, heißen "Entitäten (`entities`)". Dies können beliebige Objekte sein, z.B.: reale Gegenstände, gedankliche Einheiten, Vorgänge [16].

Es gibt zwei Arten von Relationen:

1. Entitäts-Relationen (**Entitätsmengen**)
Sie beschreiben die Eigenschaft einer Klasse von Entitäten, d.h. eines Entitätstyps.
2. "Relationship" - Relationen (**Beziehungsmengen**)
Sie verknüpfen 2 (oder mehr) Entitäten und beschreiben diese Verknüpfung mit Attributen, die nur dieser Beziehung zugeordnet werden können.

Es gibt zahreiche Varianten des Entity-Relationship Modells[17].All diese Modelle dienen in erster Linie theoretischen Überlegungen oder der Strukturierung von Da-tensammlungen (konzeptioneller Entwurf), um Verknüpfungen deutlich zu machen und danach Dateien in einem der üblichen Datenbanksysteme sinnvoll definieren und einrichten zu können.
Mit Entitäts- und Beziehungsmengen läßt sich jeder Ausschnitt der realen Welt (Miniwelt) beliebig detailliert beschreiben. Eine Beziehung kann dabei auch aus mehr als zwei Entitätsmengen bestehen.

Bsp.: Die Beziehung "Hoert" zwischen "Student", "Vorlesung" und "Professor" ist eine dreistellige Beziehung.

Das Entity-Relationship Modell stellt den Rahmen für die Angabe derartiger mehr-stelliger Beziehungen bereit (konzeptioneller Entwurf). Natürlich kennt das Entity-Relationship Modell auch die in der realen Welt häufig vorkommenden binären Relationen, z.B.:

1. Die Beziehung zwischen Ehemann und Ehefrau ist vom Typ 1 : 1.
2. "Professoren können mehr als eine Diplomarbeit (gleichzeitig) betreuen, eine Diplomarbeit muß von genau einem Professor betreut werden". Das ist eine hierarchische "1 : n"-Beziehung.

Gelegentlich stehen Entitätsmengen auch über eine "ist_ein (IS_A)"-Beziehung in Verbindung. So ist bspw. ein Professor an einer Hochschule ein "Hochschulangestellter". "Hochschulangestellte" sind aber auch die Verwaltungsangestellten. Der Professor an einer Hochschule besitzt neben der Identifikation, die ihn als Hochschulmitarbeiter ausweist und einer Reihe weiterer

[15] vgl. Chen, P., S.: The Entity-Relationship-Modell: toward a unified view of data, ACM Transactions on Database Systems, March 1976
[16] vgl. 1.3.1.1
[17] Es spielt auch eine wesentliche Rolle bei den Werkzeugen des computerunterstützten Software-Engineering (CASE)

Attribute (Name, Geburtsdatum, Adresse) Merkmale, die ihn als Mitglied eines Fachbereichs in einer bestimmten Hochschule (z.B. Fachhochschule Regensburg) ausweisen.
Die zweistellige Beziehung "E_1 ist_ein E_2" besagt: E_1 ist eine Spezialisierung von E_2 (bzw. E_2 ist eine Verallgemeinerung von E_1). Im vorliegenden Fall könnte E_1 (Professoren) auch als Teilmenge von E_2 (Hochschulmitarbeiter) verstanden werden. Die Darstellung von E_1 und E_2 als verschiedene Mengen ermöglicht es jedoch, Besonderheiten von E_1 zu modellieren, die nicht unbedingt für jede Entität aus E_2 relevant sind.

Darstellung

Das Entity-Relationship Modell dient vor allem zur Beschreibung des konzeptuellen Schemas[18] einer Anwendung. Die Struktur aus Entitätsmengen, Beziehungsmengen und Attributen wird im **E**ntity-**R**elationship-Diagramm (**ER**-Diagramm) grafisch [19] dargestellt:

1. Deklaration von Entitätsmengen
Sie erfolgt über ein Rechteck, das den Namen der Entitätsmenge enthält, und durch Kreise, die die Attribute aufnehmen. Die Kreise werden durch ungerichtete Kanten mit dem Rechteck verbunden. Elemente des Primärschlüssel werden unterstrichen, Wertebereiche werden nicht dargestellt.
2. Deklaration von Beziehungsmengen
Sie erfolgt durch eine Raute [20], die den Namen der Beziehungsmenge enthält. Die Raute wird durch Kanten mit der beteiligten Entitätsmengen-Deklaration (Recht-ecke) verbunden. Die Kanten sind nicht gerichtet (Ausnahme: Hierarchische Beziehungen). Der Typ der Beziehung wird an die Kante geschrieben. Falls die Beziehungsmenge Attribute besitzt, werden diese ebenfalls durch Kreise dargestellt und über (ungerichtete) Kanten mit der entsprechenden Raute verbunden.

Bsp.: Beziehung zwischen Lieferant und Artikel im ERM

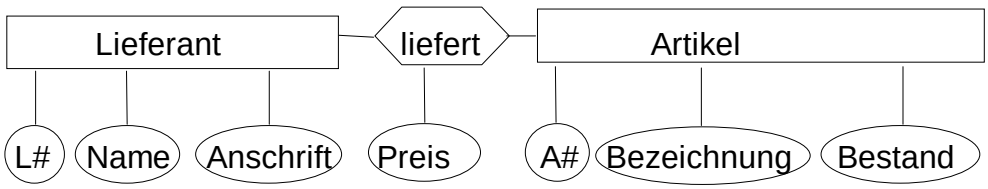


Abb. 1.3-5: ERM-Diagramm zur Beziehung "Lieferant-Artikel"

Zur Darstellung der Ausprägungen (Mengen, Occurrences) greift das ERM häufig auf das relationale Datenbankmodell zurück. Für jeden Entitätstyp und jeden Beziehungstyp wird eine eigene Tabelle angelegt. Jedes notwendige Attribut erhält in einer solchen Tabelle eine Spalte zugeordnet. Bei den Tabellen der Beziehungstypen muß darauf geachtet werden, daß auch die Primärschlüssel der über die Beziehungstypen verknüpften Entitätstypen vorliegen.

[18] vgl. 1.3.6
[19] vgl. Chen, P. S. und Knöll, Heinz-Dieter: Der Entity-Relationship-Ansatz zum logischen Systementwurf, Mannheim/Wien/Zürich, 1991
[20] Sie wird zur besseren Übersicht generell abgeflacht dargestellt

Lieferant		
L#	Name	Anschrift
L01	....	....

liefert	
L#,A#	Preis
L01,A17	.....

Artikel		
A#	Bez	Best
A17	....	....

Abb. 1.3-6: Relationale Datenbank zur Beziehung "Lieferant-Artikel"

Die Komplexität eines Beziehungstyps gibt an, in welchem Verhältnis die Entitäten der beteiligten Entitäts-Typen zueinander in Beziehung stehen. Sie wird durch die Beschriftung der Kanten (1, m, n) ausgedrückt. Zwischen 2 Entitäts-Typen können Beziehungen des Typs 1:1, 1:n (eins zu viele) und m:n (viele zu viele) vorkommen[21]. Einer Kante kann ein Rollenname zugeordnet sein, der die Funktion des jeweiligen Entitäts-Typen in Bezug auf den Beziehungstyp beschreibt.

3. „**ist_ein**“-Beziehung
Sie wird durch eine Raute mit der Beschriftung „ist_ein“ oder durch einen Pfeil dargestellt. Die Kante zu E_1 einer Beziehung "E_1 ist_ein E_2" ist ungerichtet, die Kante zu E_2 gerichtet.

Bsp.: Entity-Relationship Diagramme zu einer Hochschulverwaltung

Entitätsmengen[22] sind

```
Hochschulmitarbeiter
(MNR, Gebdat, Adr, ...)
MNR ... Mitarbeiternummer [23]

Diplomarbeit
(Thema, Beginn, Ende, Note)
Student
(Matrikelnummer, Name, Vorname, .... )
Vorlesung
(Name, ....)
Professor
(MNR, Fachbereich, Name, ... )
```

Diese Entitätsmenge besitzt eine "ist_ein" - Beziehung zu Hochschulmitarbeiter.

Beziehungsmengen sind

```
Betreut
```
("Professor - Diplomarbeit"). Der Beziehungstyp ist hierarchisch (1 : n)

[21] vgl. 1.3.1.2
[22] Identifizierende Attribute sind unterstrichen
[23] Die Mitarbeiternummer ist ein Attribut, das allen Hochschulmitarbeitern gemeinsam ist und veerbt wird

`Erarbeitet`
Der Beziehungstyp ("Student - Vorlesung - Professor") ist vom Typ n : p : m und besitzt die Attribute Semester, Jahr.
`Schreibt`
Der Beziehungstyp "Diplomarbeit - Student" ist vom Typ m : n und besitzt das Attribut Semester.

Das führt zu dem folgenden Entity-Relationship-Diagramm:

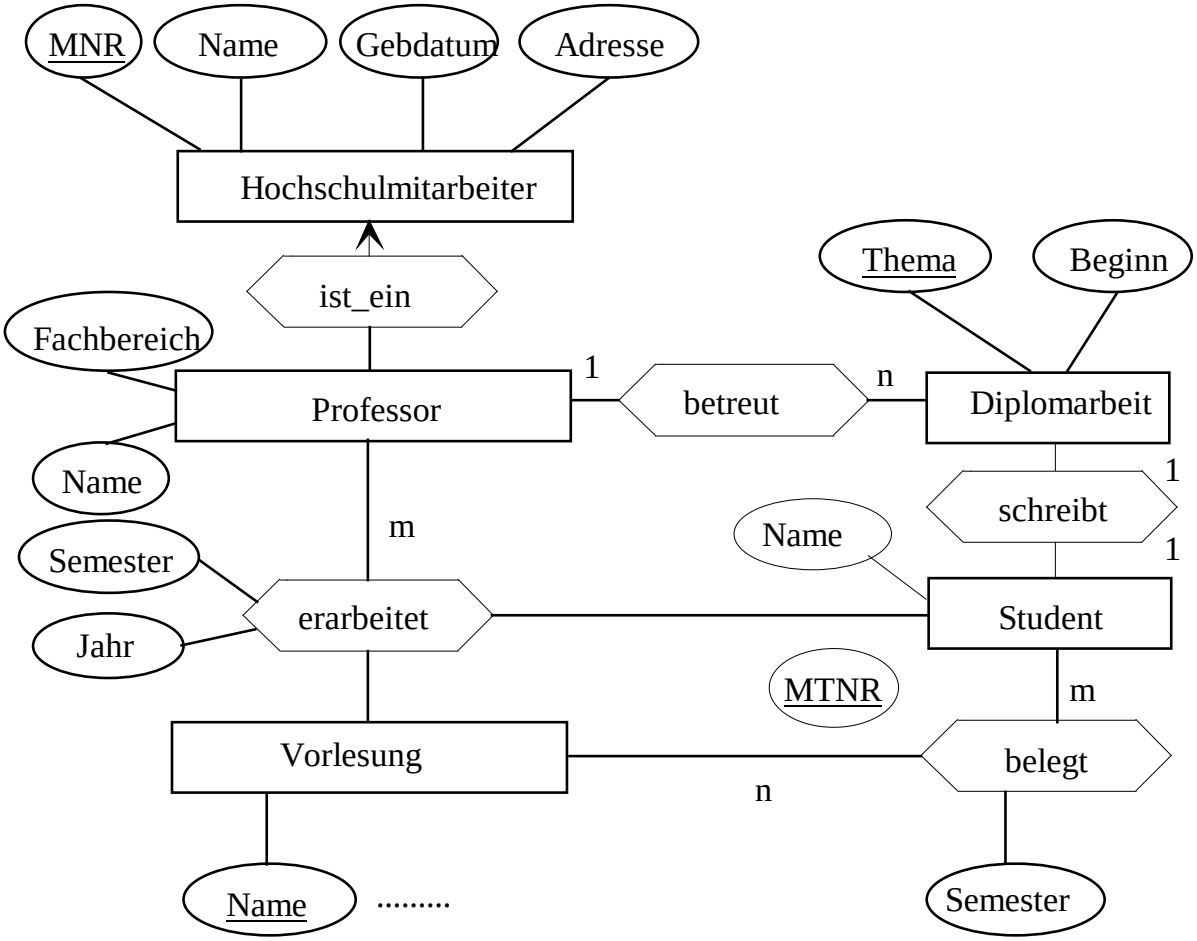
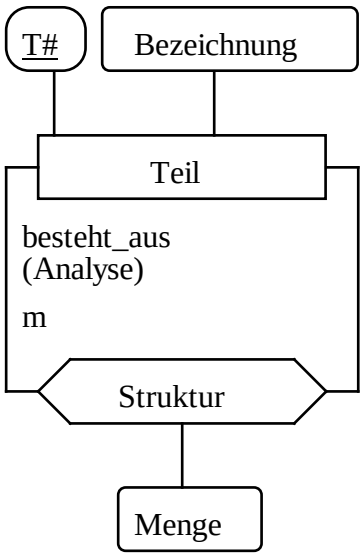


Abb. 1.3-7: Entity-Relationship-Diagramm zu einer Hochschul-Verwaltung

Homogene Zusammenstellungen von Beziehungsmengen der Klassifikation E - E sind ebenfalls im ERM darstellbar, z.B. Stückliste und Verwendungsnachweis bilden sich im ER-Diagramm so ab:



Zur Darstellung der Mengen greift das ERM auf das Relationenmodell zurück.

Teil	
T#	Bezeichnung
E1	Einzelteil-1
E2	Einzelteil_2
E3	Einzelteil_3
B1	Baugruppe_1
B2	Baugruppe_2
P1	Endprodukt_1
P2	Endprodukt_2

Struktur		
OT#	UT#	Menge
P1	B1	2
P1	B2	3
P1	E3	10
P2	B1	3
P2	E3	8
B1	E1	7
B1	E2	8
B2	E2	10
B2	E3	4

Abb. 1.3-8: Entity-Relationship-Diagramm für eine Teile-Verwaltung

"**Dreierbeziehungen**"

Die Beziehungsmenge "Student-Vorlesung-Professor (n:p:m) ist eine Dreierbeziehung (Dreifachbeziehung). Auch 1:n:m-, 1:1:n- und 1:1:1-Beziehungen sind möglich.

Bsp.: Studenten an Fachhochschulen haben zwei praktische Studiensemester zu absolvieren. Es wird gefordert, daß im Rahmen des Praktikums eine Mitwirkung des Studenten an Projektarbeiten stattfindet. Zweckmäßigerweise gilt dann:

- kein Betreuer führt einen beliebigen Studenten in mehr als einem Projekt
- kein Student arbeitet an einem beliebigen Projekt, das von mehr als einem Betreuer betreut wird.

Das ER-Modell drückt das so aus:

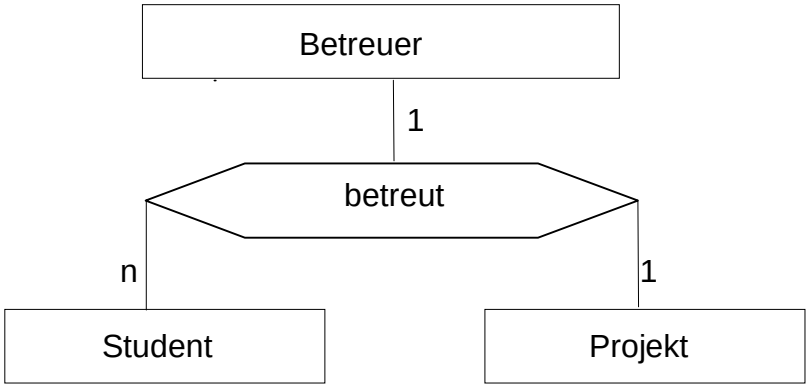


Abb. 1.3-9: Eine 1:1:n-Dreierbeziehung

„**Konstruktionsoperatoren**“

Durch Generalisierung werden ähnliche oder miteinander verwandte Objekttypen zu übergeordneten Objekttypen zusammengefaßt. In einem ER-Diagramm kann die Generalisierung durch eine „ist_ein“-Beziehung veranschaulicht werden.

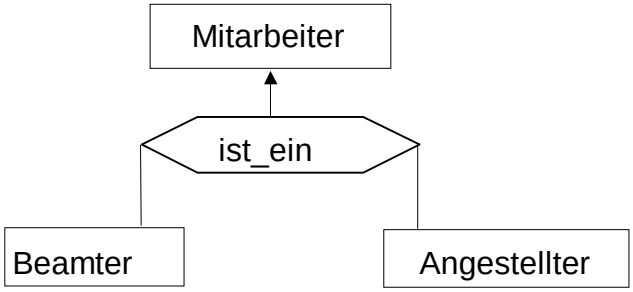


Abb. 1.3-10: ER-Diagramm zur Beschreibung der Mitarbeiter an einer FH

Spezialisierung zerlegt Objekttypen in speziell definierte nachgeordnete Entitätstypen. Spezialisierung ist die Umkehrung der Generalisierung. So ist die vorstehende Abbildung auch eine Spezialisierung, da dier Entitätstyp „Mitarbeiter“ in die Entitätstypen „Beamter“ und „Angestellter“ spezialisiert wird.
Mit der "**ist_ein**"-Beziehung werden **Untertypen** (Subtypen) in das ERM eingeführt. Ein Entitätstyp E_1 ist ein Untertyp des Entitätstyps E_2, falls jede Ausprägung (Instanz) von E_1 auch ein Untertyp von E_2 ist.

Bsp.: In einem Software-Unternehmen für Prozeßautomatisierung gibt es Sekretärinnen, Manager und Ingenieure. Die Ingenieure können Maschinenbau-, Flugzeugbau-, Bauingenieure sein. Im ER-Diagramm drückt sich das so aus:

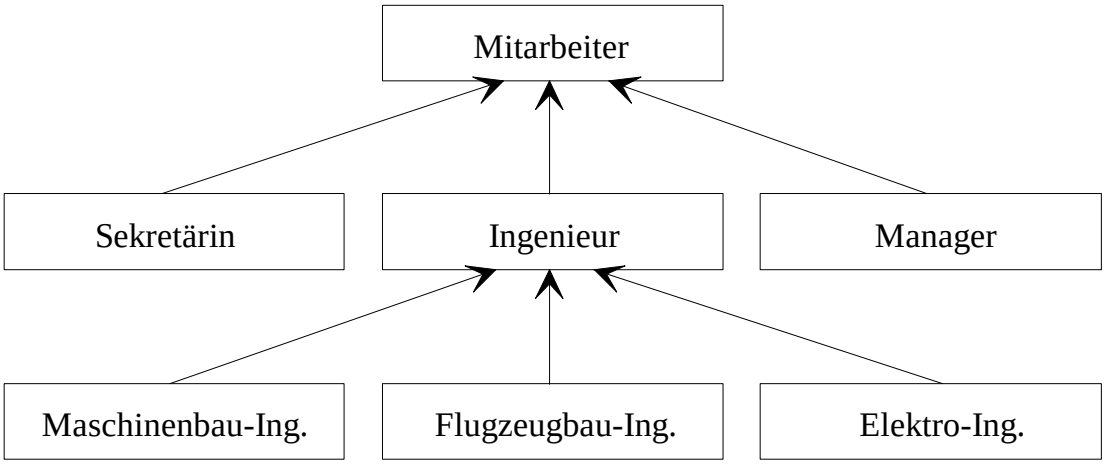
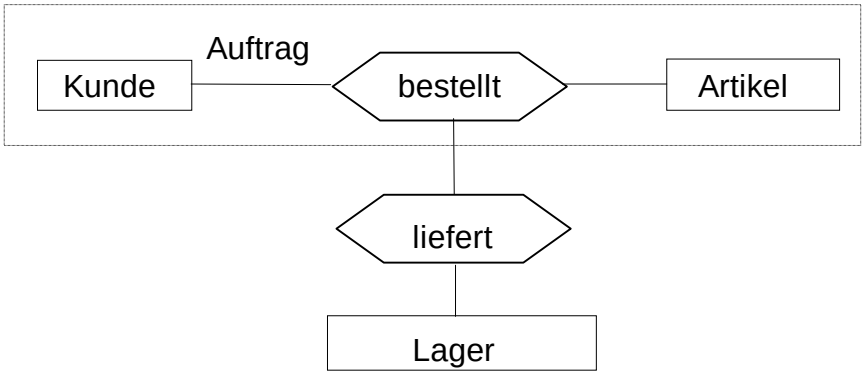


Abb. 1.3-11: Mitarbeiterbeziehungen in einer Projektgruppe

Diese Hierarchie könnte in das folgende Schema einer relationalen Datenbank überführt werden:

Mitarbeiter(MNR, (Attribute, die allen Mitarbeitern gemeinsam sind))
Ingenieure(MNR, (Attribute, spezifisch für Ingenieure))
Sekretärin(MNR, (Attribute, spezifisch für Sekretärinnen, Sekretäre))
Maschinenbau-Ingenieure(MNR, (Attribute, spezifisch für Maschinenbau-Ingenieure))
Flugzeugbau-Ingenieure(MNR, (Attribute, spezifisch für Flugzeugbau-Ingenieure))
Elektro-Ingenieure(MNR, (Attribute, spezifisch für Elektro-Ingenieure))
Manager(MNR, (Attribute, spezisch für Manager))

Entitäten, die Mitglieder eines Untertyps sind, erben die Attribute ihrer übergeordneten Typen. Untertypen könen zusätzliche spezifische Attribute und Beziehungen haben.
Durch Aggregation werden Beziehungen und zugehörige Entitäten zu Entitäten auf höherer Ebene zusammengefaßt. So kann in dem folgenden ER-Diagramm „Kunde“, „Artikel“ und der Beziehungstyp „bestellt“ zusammengefaßt werden.



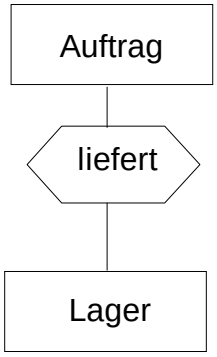


Abb. 1.3-12: : Aggregation

Schwache Entitätstypen

Manchmal ist es nicht möglich, eine Entität des Typs E anhand der Ausprägungen seiner Attribute zu identifizieren. In solchen Fällen wird eine Entität E über die Beziehungsmenge E-E' identifiziert. E ist dann ein schwacher Entitätstyp. Jeder Beziehungstyp, der mit einem schwachen Entitätstyp verknüpft ist, ist ein schwacher Beziehungstyp. Im ER-Diagramm wird ein schwacher Entitätstyp durch ein doppelt umrahmtes Rechteck, die Richtung der Abhängigkeit durch einen Pfeil symbolisiert, z.B.:

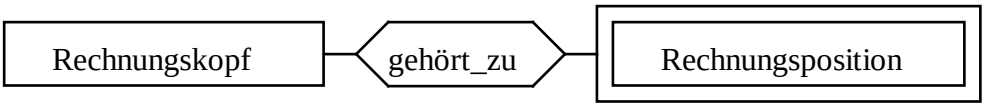


Abb. 1.3-13: Entity-Relationship-Diagramm für eine Rechnung

Durch das Konzept des schwachen Entitäts-Typs werden Existenzabhängigkeiten in das ERM eingeführt. In dem vorliegenden Beispiel hängt die Existenz einer Rechnungsposition von der Existenz des zugehörigen Rechnungskopfes ab. Es besteht eine Schüsselabhängigkeit zu einer anderen Entität. Beziehungstypen, die diese Identifizierung herbeiführen, werden häufig auch im ER-Diagramm in einer Raute mit doppeltem Umriß gezeichnet.

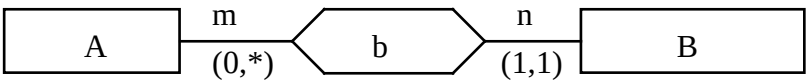
Erweiterungen des ERM

Zum ERM wurde eine Vielzahl von Varianten und Erweiterungen vorgeschlagen. **Eine** sehr sinnvolle **Erweiterung** bezieht sich auf die Präzisierung der Komplexität von Beziehungen. Die bisher angegebene Darstellung der Kompexität von Be-ziehungen ist durch das Verhältnis zwischen 2 Entitäten bestimmt. Eine "1:n"-Be-ziehung zwischen z.B. einem Kunden und einer Rechnung sagt aber nichts darüber aus, ob jedem Kunden wenigstens eine Rechnung zugeordnet sein muß oder nicht. Auch ist nicht ersichtlich, ob sich die Rechnung genau auf einen Kunden bezieht oder ob Rechnungen ohne Kunden zulässig sind. Bei drei- und mehrstelligen Beziehungen ist die "(1,m,n)"-

Schreibweise überhaupt nicht sinnvoll interpretierbar. Wird für jeden Entitätstyp durch einen Komplexitätsgrad comp(E,b) angegeben, mit wievielen Beziehungen des Typs b eine Entität minimal in Beziehung stehen muß bzw. maximal in Beziehung stehen kann, dann liegt die "(min,max)"-Schreibweise vor. Es gilt

0 <= min <= 1 <= max <= * [24]

Eine Beziehung b(A,B) wird durch 2 Komplexitätsgrade comp(A,b) und comp(B,b) beschrieben, z.B.:



Der als Zahlenpaar angegebene Komplexitätsgrad bedeutet:

- Die erste Zahl vor dem Komma gibt die Mindestzahl an
- Die zweite Zahl nach dem Komma gibt die Höchstzahl an. „*“ bedeutet viele.

Eine Beschreibung des Beziehungstyps b(A,B) kann durch die Komplexitätsgrade comb(A,b) bzw. comp(B,b) allgemein so erfolgen:

b(A,B)	comp(A,b)	comp(B,b)
1:1	(0,1) oder (1,1)	(0,1) oder (1,1)
1: n	(0,*) oder (1,*)	(0,1) oder (1,1)
n:1	(0,1) oder (1,1)	(0,* oder (1,*)
n:m	(0,*) oder (1,*)	(0,*) oder (1,*)

Abb. 1.3-14: Komplexitätsgrade im ERM

Bsp.: Ein typisches ER-Diagramm aus der Fertigungsvorbereitung

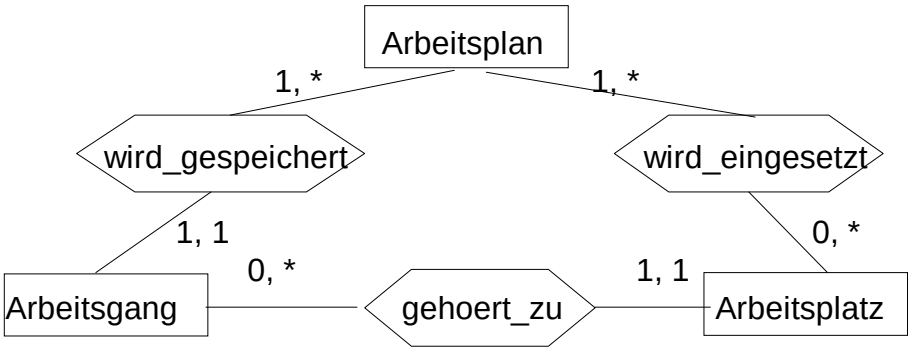


Abb. 1.3-15: ER-Diagramm aus der Fertigungsvorbereitung

[24] * bedeutet beliebig viele

Zusammenfassung: „Arbeitsschritte beim Ausarbeiten von Datenstrukturen in einem ER-Diagramm“

(1) Ermittle relevante Entitäten für den betrachteten Datenbereich
(2) Feststellen der Entitätsmengen. Nur gleichartige Entitäten können zur Entitätsmenge zusammengefaßt werden.
(3) Ermitteln der Attribute für jede Entitätsmenge (evtl. mit Korrekturen des 2. Arbeitsschrittes)
(4) Definition der Beziehungstypen von Entitätsmengen, zwischen denen Beziehungen bestehen.
(5) Ermitteln der Häufigkeiten der Beziehungen zwischen den Entitätsmengen
(6) Darstellung der Datenstruktur in einem ER-Diagramm

Aufgabe: „Entwerfe das ER-Diagramm, das Schema und die relationale Datenbank für die Personalverwaltung in einem Unternehmen".

a) Entwurf des ER-Diagramms

Die Analyse zur Ermittlung der Struktur einer relationale Datenbank hat ergeben:

(1) Ermitteln relvanter Entitäten für den betrachteten Datenbereich, z.B.:

- Der Angestellte "A1" mit dem Namen "Fritz" ist am "2.1.1950" geboren und arbeitet in der Abteilung "OD (Organisation und Datenverarbeitung)" als "Systemplaner (SY)". Er besitzt zusätzliche Qualifikationen für "Operateur (OP)-", "Programmierer (PR)"-Tätigkeiten. Ein "Systemplaner" verdient "6000.00.-DM", ein "Operateur (OP)" "3500.-DM", ein "Programmierer" 5000.00.-DM" im Monat.
- Der Angestellte "A2" mit dem Namen "Tom" ist am "2.3.1951" geboren und arbeitet in der Abteilung "KO (Konstruktion)" als "Ingenieur (IN)". Er besitzt zusätzliche Qualifikation zur "Systemplaner (SY)-"Tätigkeiten. Ein "Systemplaner" verdient "6000.00.-DM", ein "Ingenieur (IN)" "6000.- DM" im Monat.
- Der Angestellte "A3" mit dem Namen "Werner" ist am "23.1.1948" geboren und arbeitet in der Abteilung "OD (Organisation und Datenverarbeitung)" als "Programmierer (SY)". Er besitzt zusätzliche Qualifikationen für "Operateur (OP)-", "Programmierer (PR)"-Tätigkeiten. Ein "Programmierer" verdient "5000.00.-DM" im Monat.
- Der Angestellte "A4" mit dem Namen "Gerd" ist am "3.11.1955" geboren und arbeitet in der Abteilung "VT (Vertrieb)" als "Kaufm. Angestellter (KA)". Ein "Kaufm. Angestellter" verdient "3000.00.-DM" im Monat.
- Der Angestellte "A5" mit dem Namen "Emil" ist am "2.3.1960" geboren und arbeitet in der Abteilung "PA (Personalabteilung)" als "Programmierer (PR)". Ein "Programmierer" verdient "5000.00.-DM" im Monat.
- Der Angestellte "A6" mit dem Namen "Uwe" ist am "3.4.1952" geboren und arbeitet in der Abteilung "RZ (Rechenzentrum)" als "Operateur (OP)". Ein "Operateur (OP)" verdient "3500.- DM", ein "Programmierer" im Monat.
- Die Angestellte "A7" mit dem Namen "Erna" ist am "17.11.1955" geboren und arbeitet in der Abteilung "KO (Konstruktion)" als "Techn. Angestellte (TA)". Eine "Techn. Angestellte" verdient "3000.00.-DM" im Monat.
- Die Angestellte "A8" mit dem Namen "Rita" ist am "2.12.1957" geboren und arbeitet in der Abteilung "KO (Konstruktion)" als "Techn. Angestellte (TA)". Eine "Techn. Angestellte" verdient "3000.00.-DM" im Monat.
- Die Angestellte "A9" mit dem Namen "Ute" ist am "8.9.1962" geboren und arbeitet in der Abteilung "OD (Organisation und Datenverarbeitung)" als "Systemplaner (SY)". Sie besitzt zusätzliche Qualifikation für "Ingenieur (IN)"-Tätigkeiten. Ein "Systemplaner" verdient "6000.00.-DM", ein "Ingenieur (IN)" "6000.- DM" im Monat.
- Der Angestellte "A10" mit dem Namen "Willi" ist am "7.6.1956" geboren und arbeitet in der Abteilung "KO (Konstruktion)" als "Ingenieur (IN)". Ein "Ingenieur" verdient "6000.00.-DM" im Monat.

- .....

(2) Daraus lassen sich die folgendenen Entitäten ableiten

- Abteilung, Angestellte, Job

(3) , die folgende Attribute besitzen:

- Abteilung(Abt_ID,Bezeichnung)
- Angestellte(Ang_ID,Name,Gebjahr,Abt_ID,Job_ID)
- Job(Job_ID,Titel,Gehalt)

(4) Beziehungen bestehen zwischen

Abteilung und Angestellte
Angestellte und Job (ausgeübeter Beruf)
Angestellte und Job (Qualifikation)

(5) Ermitteln der Häufigkeit von Entitätsmengen zwischen denen Beziehungen bestehen:

Ein Angestellter arbeitet in einer Abteilung.
Eine Abteilung hat mehrere Angestellte.
Ein Angestellter übt eine bestimmte berufliche Tätigkeit aus.
Eine berufliche Tätigkeit kann von mehreren Angestellten wahr genommen werden.
Ein Angestellter besitzt mehrere Qualifikationen,
Eine bestimmte Qualifikation haben mehrere Angestellte erreicht.

(6) Darstellung der Datenstruktur in einem ER-Diagramm

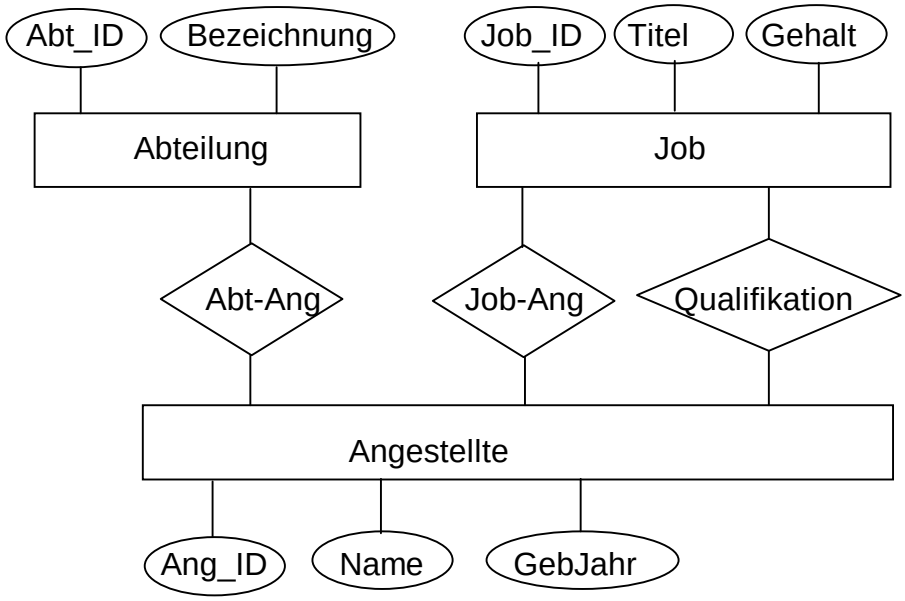


Abb. 1.3-16: ER-Diagramm zur Datenbank Personalverwaltung

b) Schemaentwurf der reltionalen Datenbank

- Abteilung(Abt_ID,Bezeichnung)
- Angestellte(Ang_ID,Name,Gebjahr,Abt_ID,Job_ID)
- Job(Job_ID,Titel,Gehalt)
- Qualifikation(Ang_ID,Job_ID)

c) Die relationale Datenbank für das Personalwesen

ABTEILUNG

ABT_ID	BEZEICHNUNG
KO	Konstruktion
OD	Organisation und Datenverarbeitung
PA	Personalabteilung
RZ	Rechenzentrum
VT	Vertrieb

ANGESTELLTE

ANG_ID	NAME	GEBJAHR	ABT_ID	JOB_ID
A1	Fritz	2.1.1950	OD	SY
A2	Tom	2.3.1951	KO	IN
A3	Werner	23.4.1948	OD	PR
A4	Gerd	3.11.1950	VT	KA
A5	Emil	2.3.1960	PA	PR
A6	Uwe	3.4.1952	RZ	OP
A7	Eva	17.11.1955	KO	TA
A8	Rita	02.12.1957	KO	TA
A9	Ute	08.09.1962	OD	SY
A10	Willi	7.7.1956	KO	IN
A11	Erna	13.10.1966	OD	KA
A12	Anton	5.7.1948	OD	SY
A13	Josef	2.8.1952	KO	SY
A14	Maria	17.09.1964	PA	KA

JOB

JOB_ID	TITEL	GEHALT
KA	Kaufm. Angestellter	3000,00 DM
TA	Techn. Angestellter	3000,00 DM
SY	Systemplaner	6000,00 DM
PR	Programmierer	5000,00 DM
OP	Operateur	3500,00 DM

QUALIFIKATION

ANG_ID	JOB_ID
A1	SY
A1	PR
A1	OP
A2	IN
A2	SY
A3	PR
A4	KA
A5	PR
A6	OP
A7	TA
A8	IN
A9	SY
A10	IN
A11	KA
A12	SY
A13	IN
A14	KA

Abb. 1.3-17: Tabellen zur relationalen Datenbank

1.3.4 Das netzwerkorientierte Datenbankmodell

Das Datenmodell einer netzwerkorientierten DB sieht vor, daß die gespeicherten Objekte beliebig miteinander verknüpft sein können. Durch dieses Datenmodell werden Objekttypen und die vielfältigen Beziehungen, die zwischen Objekttypen bestehen können, beschrieben.

1. Beziehungen

Die Beziehungen zwischen den Objekttypen haben die Form einfacher Netzwerkstrukturen und tragen eine Benennung. In einer grafischen Darstellung sind das Pfeile, die die in eckigen Kästchen dargestellten Objekttypen (Entitäts-, Satztypen) verbinden. Die Benennung der Beziehungstypen (Set-Typen) nimmt ein Oval (Ellipse) auf. Eine Beziehung, z.B. zwischen den Entitätstypen E bzw. E' stellt sich dann folgendermaßen dar:

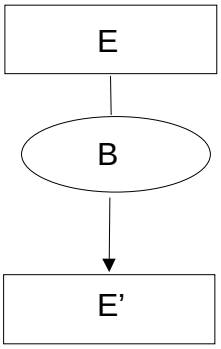


Abb. 1.3-20: Datenbankstrukturdiagramm im Netzwerkmodell

Beschreibt man in dieser Form einen umfassenden Bereich (z.B. alle Daten eines Unternehmens), so erhält man einen gerichteten Graphen:

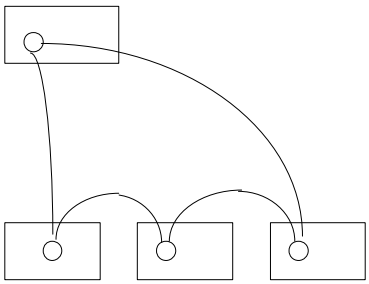
- Seine Knoten sind Entitätstypen
- Seine Kanten sind Beziehungen zwischen den Entitätstypen, da alle möglichen Beziehungen zugelassen sind, erhält man ein Netzwerk (plex structure).

2. Definition des Set-Typ

Jede Entität e' vom Typ E' steht höchstens mit einer Entität e vom Typ E in Bezie-hung.
Zu jeder Entität e vom Typ E gehört ein Set vom Set-Typ B. Neben e gehören dazu alle Entitäten e_1',e_2', ... , e_k' vom Typ E'.
Ein Set-Typ bestimmt eine "1 : n"-Beziehung.
Die Ausprägungen eines speziellen Set-Typs beschreibt man mit Hilfe eines Occurrence-Diagramms:

Nichtleerer Set

Owner



Member

Abb. 1.3-21: Occurrence-Diagramm im Netzwerkmodell

Leerer Set: Der Owner besitzt keine Member

3. Die Datenbank des Netzwerk-Datenmodell

- Es gibt eine Menge von benannten Entitäts-Typen und eine Menge von benannten Set-Typen
- Jede Entität der Datenbank gehört genau zu einem Entitäts-Typ, jede konkrete Beziehung zwischen Entitäten zu genau einem Set-Typ
- Den Set-Typen sind keine Attribute zugeordnet
- Die Member jedes Set sind (gemäß einer dem Set-Typ zugeordneten Ordnung) geordnet.
- Eine Teilmenge von Entitäts-Typen ist ausgezeichnet. Für sie gilt: Zu jeder Entität e der Datenbank gibt es mindestens einen (gerichteten) Weg $e_0,e_1,e_2,....,e_{k-1},e_k$,dessen Ausgangspunkt e_0 von einem Typ E_0 ist. Entitäten von einem Typ E_0 heißen Einstiegpunkte in das Netzwerk.
- Jede Entität, die nicht von einem Typ E_0 ist, kann nur über eine vom Einstieg-punkt ausgehende Folge von Sets erreicht und dem Benutzer zugänglich gemacht werden. Dabei kann ein Set auch vom Member zum Owner durchlaufen werden.

Bsp.: Komplexe Beziehungen zwischen Angestellten und Projekten

Abb. 1.3-22: ER-Diagramm für "Angestellter" und "Projekt"

"A_P" ist hier vom Typ m:n, denn

- ein Angestellter arbeitet an mehreren Projekten bzw.
- ein Projekt wird von mehr als einem Angestellten bearbeitet

z.B.: Es bestehen zwischen den 4 Angestellten A1, A2, A3, A4 und den Projekten P1, P2, P3 die folgenden konkreten Beziehungen:

	P1	P2	P3
A1	x	x	
A2		x	
A3		x	
A4	x		x

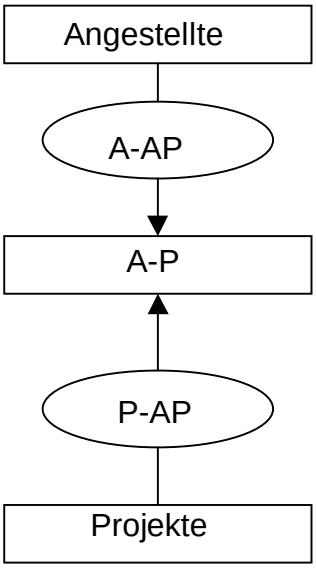
Realisierung im Netzwerk-Modell?

Notwendig ist die Einführung eines neuen, der Beziehung A_P entsprechenden Entitäts-Typ E_B (z.B. A-P). Zu jeder konkret auftretenden Beziehung wird festgelegt:

- Eine Entität (Ai-Pj) vom Typ E_B
- 2 Set-Typen b_1 (A-AP) und b_2 (P-AP)

Die neue Entität A-P kann jetzt auch Attribute zugeordnet bekommen (,z.B.: % der Arbeitszeit)

Datenbankstruktur-Diagramm



Occurrence-Diagramm

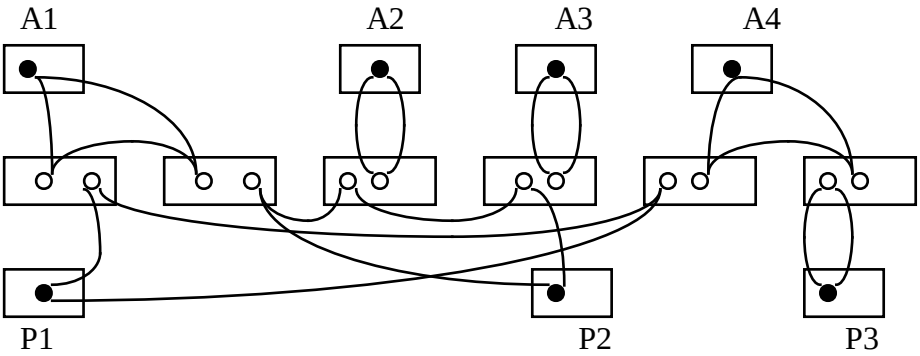


Abb. 1.3-23: Datenbankstruktur- und Occurrence-Diagramm zur Beziehung "Angestellter" und "Projekt"

4. Zugriff auf eine Entität

Von einem Einstiegpunkt in die Datenbank gibt man einen genauen Weg in der Form einer Folge von Sets an, über die man die gewünschte Entität erreichen kann. Der Benutzer muß den Zugriffspfad für jede Entität selbst angeben (Navigieren durch die Datenbank).

Bsp.: Zugriff zu den Angestellten (im Rahmen der komplexen Beziehungen zwischen Angestellten und Projekten)

Aufgabe: Finde alle Angestellte, die an Projekt P1 arbeiten!
Lösungsschritte:

1. Suche Entität P1
2. Deute P1 als Owner eines Set vom Typ P-AP
3. Suche 1./nächstes Member Ai-P1 in diesem Set
4. Falls nicht vorhanden: STOP (evtl. weiter mit 8.)
5. (Vorhanden). Deute Ai-P als Member in einem Set vom Typ A-AP, suche zugehörigen Owner Ai
6. Verarbeite Ai
7. Weiter bei 3.
8. ...

5. Zusammenfassung

Die Einführung spezieller Entitäts-Typen zur Realisierung von Beziehungen ist in folgenden Fällen notwendig:
1. Die Beziehungen sind keinem Set-Typ zuzuordnen ("n:m"-Beziehungen)
2. Die Beziehung besteht zwischen mehr als 2 Entitäts-Typen
3. Eine Beziehung besitzt Attribute

Das Schema einer auf dem Netzwerk-Datenmodell beruhenden Datenbank muß also enthalten:

- Die Beschreibung der vorhandenen Entitäts-Typen und ihrer Attribute (einschl. der Wertebereiche)
- Die Beschreibung der vorkommenden Set-Typen

Bsp.:

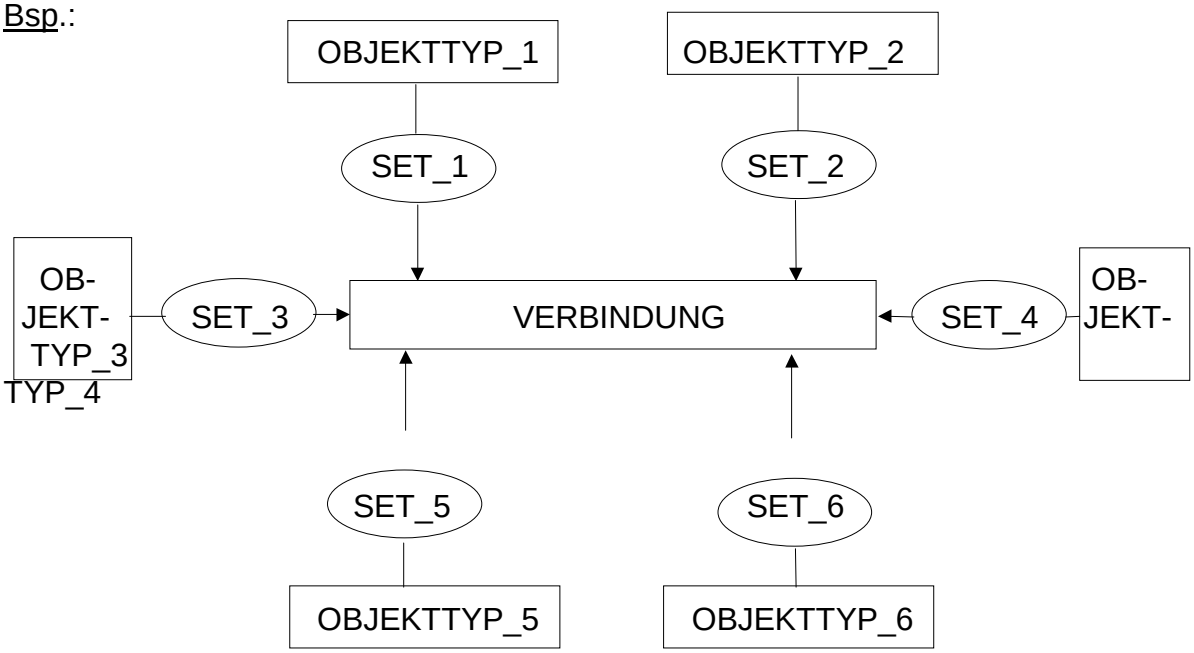


Abb. 1.3-24: Datenbankstrukturdiagramm eines Netwerks mit 6 Sets

1.3.5 Das hierarchische Datenbankmodell

Berücksichtigt werden hierarchische Beziehungen zwischen den Daten. Eine hierarchische Struktur läßt sich in der Gestalt eines Baumes beschreiben. Ein Baum ist eine Datenstruktur mit folgenden Eigenschaften:

- Sie ist zyklenfrei.
- Es gibt einen speziell hervorgehobenen Knoten, die Wurzel. Sie hat keinen Vorgänger.
- Auf jeden anderen Knoten zeigt eine Kante, d.h.: Jeder Knoten hat genau einen Vorgänger.

1. Definition

- Es gibt eine Menge benannter Entitäts-Typen (Segmente) und eine Menge von unbenannten Beziehungen.
- Jede Entität (Segment Occurrence) gehört zu einem Entitäts-Typ.
- Alle Entitäten innerhalb der Datenbankstruktur (,der zugehörige Graf ist eine Ansammlung von Bäumen,) sind total geordnet (hierarchische Ordnung)
- Jede Entität ist nur über einen Einstiegpunkt (Wurzelsegment) erreichbar.

Bsp.:

Datenbankstruktur-Diagramm

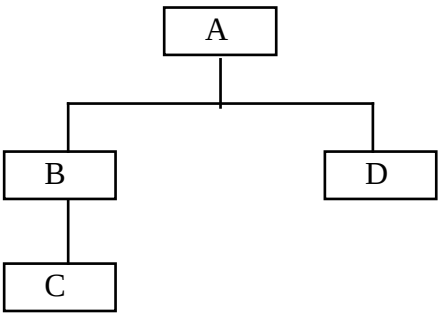
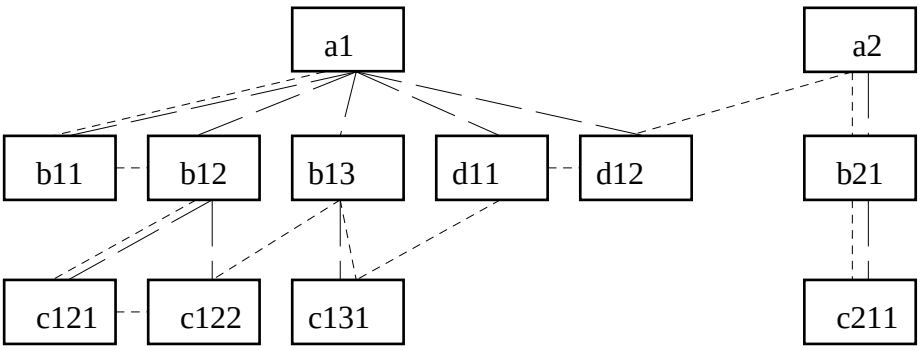


Abb. 1.3-25: Datenbankstruktur im hierarchischen Datenmodell

Occurrence-Diagramm



---------- Weg der hierarchischen Ordung

Abb. 1.3-26: Ausprägung hierarchischer Strukturen

2. Wesentliche Einschränkungen gegenüber dem Netzwerkmodell

- Nur Baumstrukturen, keine allgemeine Netzstrukturen sind erlaubt
- Zugriff kann nur über einen gerichteten Weg, der von der Wurzel ausgeht, erfolgen. Nur in einem Baum kann ein Entitäts-Typ vorkommen Ist bspw. die Beziehung E - E' in einem Baum realisiert, ist e' nur über die zugehörige Entität e vom Typ E zu erreichen. Eine selbstständige Verarbeitung von Entitäten des Typs E' ist nur so möglich: Duplizieren aller zu diesem Entitäts-Typ zugehörigen Entitäten (Redundanz!)
- Beziehungen zwischen Entitäts-Typen (Segmenten) sind nur in einer Baum-struktur zu realisieren. Beziehungstypen gibt es daher nur in der Form `1 : 1` bzw. `1 : m`.

Segmente der obersten Hierarchiestufe heißen **Wurzelsegmente**. Ein von einem solchen Segment ausgehende Hierarchie ist ein Datenbanksatz. Die Menge der von den Wurzelsegmenten ausgehenden Sätze bilden eine Datenbasis. Ein Segment ist nur über einen gerichteten Weg erreichbar, dessen Ausgangspunkt das Wurzelsegment ist.

3. Beispiele zur Anwendung hierarchischer Strukturen

a) heterogene Strukturen (`E - E'`)

Bsp.:

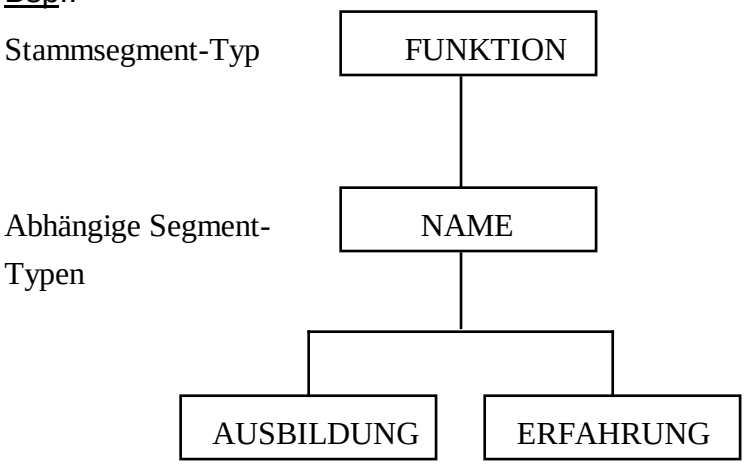


Abb. 1.3-26: Datenbankstrukturdiagramm für eine heterogene Struktur

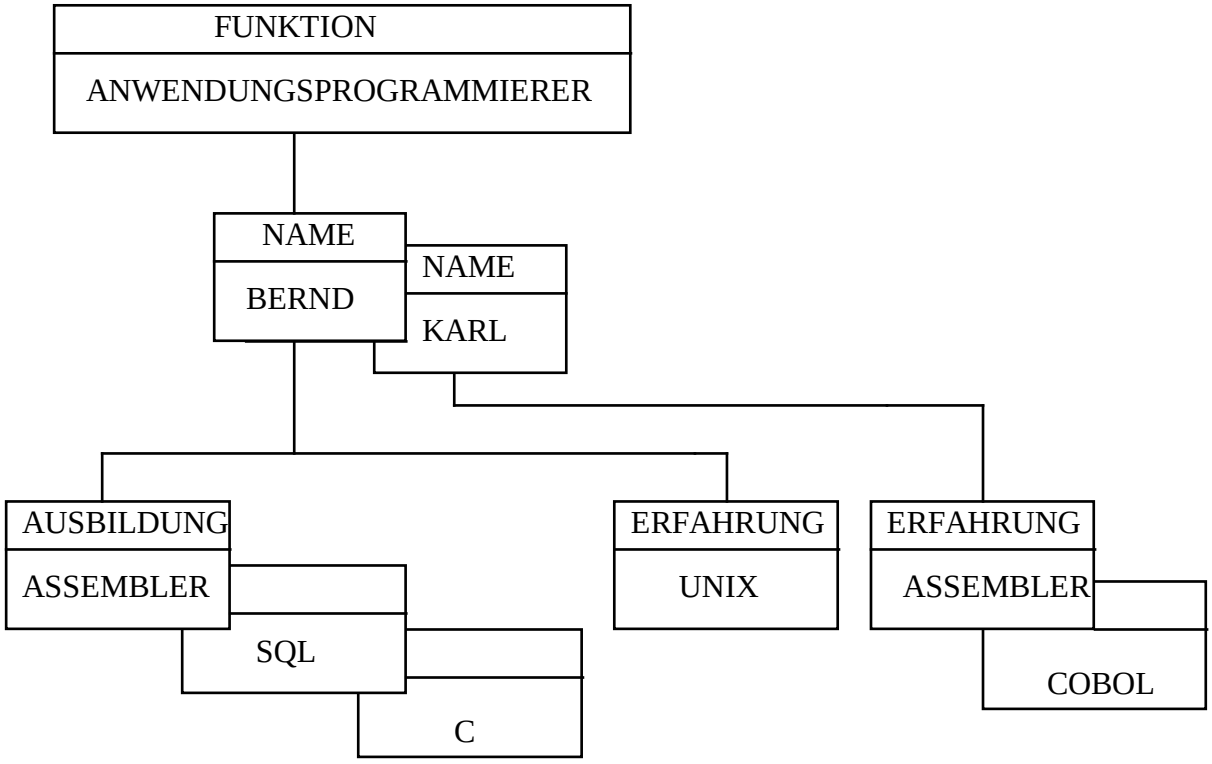


Abb. 1.3-27: Occurrence-Diagramm einer heterogenen Struktur

b) Homogene Strukturen (E - E)

Bsp.:

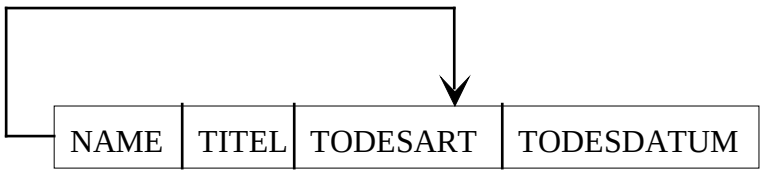


Abb. 1.3-28: Struktur einer Herschaftsfolge

Diese Schemadarstellung beschreibt bspw. die Folge deutscher Könige im Mittelalter. Die Herrscher wurden gewählt, häufig gab es sogar Gegenkönige (1:n Beziehung der Nachfolge). In einer physischen Datenbank des hierarchischen Datenbankmodells sind aber nur 1:1 - bzw. 1:n - Verbindungen zu realisieren. Es muß also zu der vorliegenden Struktur eine Ergänzung (, ein "Zeiger-Segment") geben, damit die Forderung nach einer 1:n Verbindung erfüllt werden kann. Das Zeigersegment ist mit der Zwischensatzart im Netzwerk-Datenbankmodell vergleichbar.

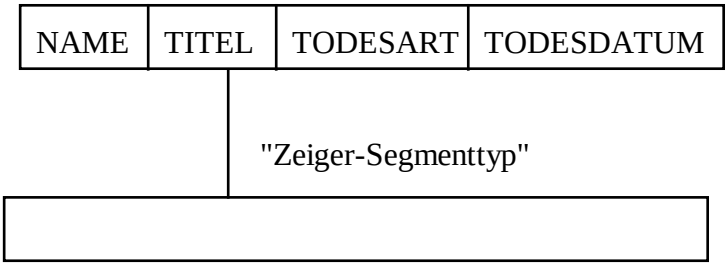


Abb. 1.3-29: Datenbankstrukturdiagramm zur Darstellung einer Herrschaftsfolge

4. Pfadabhängigkeiten

Die auf tieferen Ebenen vorkommenden Segmente sind ohne ihre Vorgänger-Segmente unvollständig:

Bsp.: Gegeben ist folgende Datenbankstruktur:

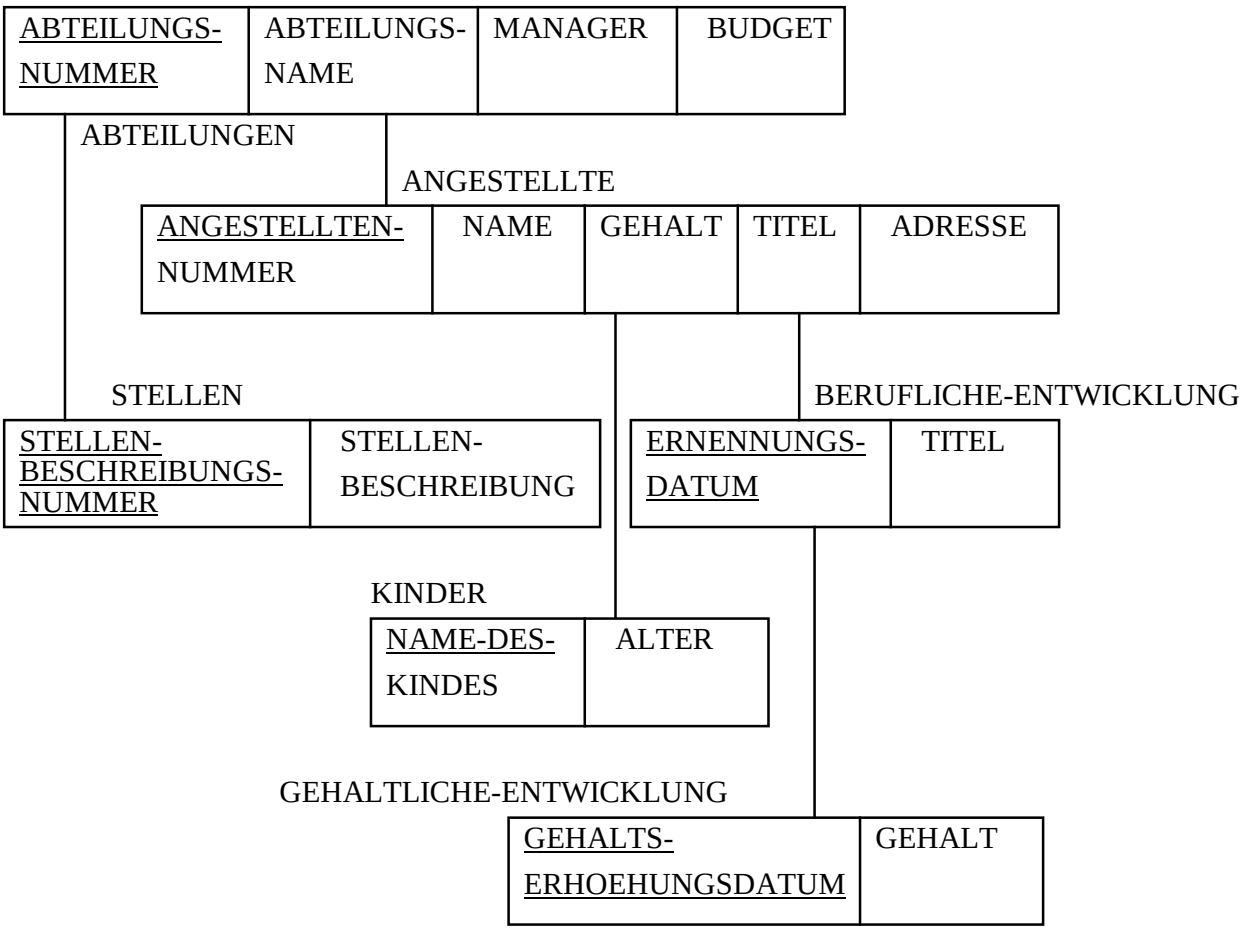


Abb. 1.3-30: Datenbankstrukturdiagramm mit Pfadabhängigkeiten

- Das Segment BERUFLICHE-ENTWICKLUNG ist ohne Bezug auf das Segment ANGESTELLTE bedeutungslos.
- Das Segment GEHALTLICHE-ENTWICKLUNG ist nicht nur vom Vatersegment sondern sogar vom Großvater-Segment ANGESTELLTE abhängig. Eine Schlüsselkombination von ANGESTELLTER und ERNENNUNGSDATUM ist zur Identifizierung des Feldes GEHALT

notwendig. Die Schlüsselkombination ABTEILUNGS-NUMMER und STELLEN-BESCHREIBUNGS-NUMMER identifiziert die STELLENBESCHREIBUNG

5. Verbindungen bei hierarchischen Datenbanken (IMS-DL/1)

Sie werden durch Zeiger realisiert. Das Segment, auf das verwiesen wird, heißt "logisches Kind (**logical child**)". Es ist i.a. kein Träger von Daten, sondern dient als Zeigersegment (auf den "logischen Vater (**logical parent**)"). Ein "**logical child**" kann nur als abhängiges Segment auftreten, das zur Vermeidung redundanter Speicherung dient. Logische Elternsegmente können mehrere logische Kinder besitzen. Ein logisches Elternsegment kann aber nicht zugleich auch "logisches Kind" eines "logischen parent" sein.

Zugriff zu Zeigersegmenten

Der Zugriff zu einem Zeigersegment kann über 2 verschiedene Wege erfolgen:

1. Der physische "parent" ist Ausgangspunkt der Abfrage
2. Der logische "parent" ist Ausgangspunkt der Abfrage

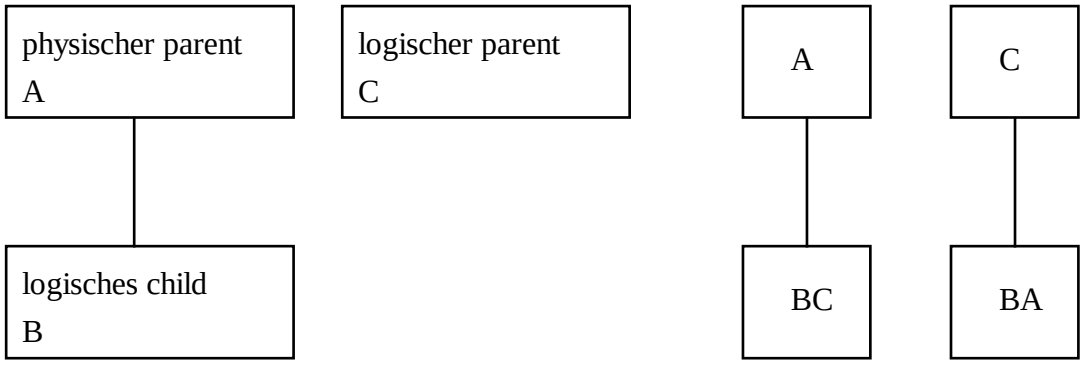


Abb.1.3-31 : Physischer und logischer Parent in hierarchischen Datenbanken

Dem Anwender werden also nicht nur die Daten des log. "parent" zur Verfügung gestellt, wenn er das "log. child" abfragt, sondern eine Kombination (concatenated segment) der Daten des "log. Kindes" und des Segments, das nicht als Ausgangs-punkt der Anfrage diente.

6. Logische Datenbankstrukturen

Ausgehend vom "Root-Segmenttyp" einer (logisch verknüpften) phy-sischen Datenbank kann jedes Segment unverändert übernommen werden, mit Ausnahme der "logical child" -Segmente. Statt dessen wird ein neuer Segmenttyp definiert, der aus der Verkettung vom "logical child" und zugehörigem "logical parent" besteht (Übergang zur logisch verknüpften Datenbank). Die dem "logical parent"-Segment über- / untergeordneten Segmente können in der logischen Datenbank als abhängige Segmente des "logical parent" definiert werden. Dabei wird die physische Datenstruktur teilweise invertiert.

1.3.6 Das Koexistenz-Modell

Datenbankarchitekturen bieten im Rahmen des sog. Koexistenzmodells drei Ebenen (Plattformen) für die Beschreibung von Daten an:

1. **Die Ebene der logischen Gestaltung**
   (Dateikonzept des jeweiligen Programmierers, externes Schema)
   Die in Anwenderprogrammen benötigten Daten werden hier (in einem Subschema) beschrieben und bereitgestellt.
2. **Die Ebene der funktionalen Gestaltung**
   Beschrieben wird hier das allumfassende Konzept für die Anwendung der Daten aus der Datenbank (konzeptuelles Schema)
3. **Die Ebene der physischen Gestaltung**
   Beschrieben wird hier die zweckmäßige Ablage der Daten auf peripheren Speichern (internes Schema).

Bsp.: „Beschreibung der Zusammenhänge bei der maschinellen Abwicklung des Bestellwesens“

- Ausgangspunkt ist die BESTELLUNG
  (Bestell-Nr.,Lieferanten-Nr.,Bestell-Datum, Liefertermin,
  Preis (der bestellten Waren (Summe))
- Der Bestellsatz ist unvollständig. Er ist zu ergänzen durch Angabe von Bestell-positionen (Wiederholungsgruppen-Problem!).
  (BESTELLPOSITION)
- Weiterhin möchte man mehr über den Lieferanten wissen!
  (LIEFERANT)
  (Lieferant-Nr.,Name (des Lieferanten),Adresse (des Lieferanten),Information (über den Lieferanten))
- Der Lieferant kann bestimmte Waren (repräsentiert durch eine Teile-Nr.) zu be-stimmten Konditionen (Preis, Liefertermin etc.) liefern.
  (PREISNOTIERUNG)
  (Teile-Nr.,Einzelpreis,Liefertermin, ...)
- Das Teil ist noch zu beschreiben (TEILE)
  (Teile-Nr.,Beschreibung, Vorrat (an Teilen im Lager)
- Schließlich ist der Teilestammsatz noch mit den vorhandenen Bestellungen (Bestell-Nr) zu verknüpfen. Es muß bekannt sein, welche Bestellungen zu einem Teil bereits vorliegen, bei dem der Vorrat zur Neige geht)

Schema

Der folgende Schema-Entwurf bietet sich an

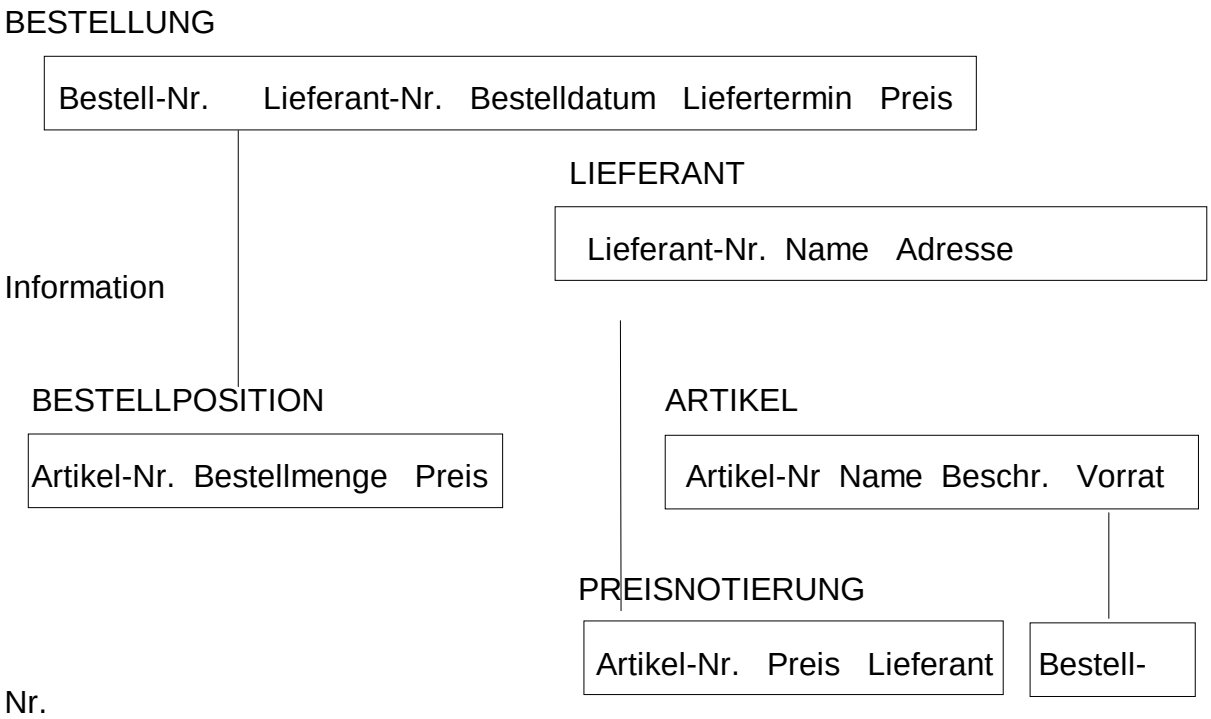


Abb. 1.3-32 : Schema zum maschinellen Abwicklung des Bestellwesens

Der Schema-Entwurf resultiert aus den unmittelbaren Beziehungen zwischen den Objekt- bzw. Satztypen. Außerdem gibt es noch Querbeziehungen. Es handelt sich dabei um ergänzende, nicht unbedingt für den unmittelbaren Verwendungszweck notwendige Informationen.

Subschema

Das Schema ist die allumfassende Darstellung der Datenelemente und Satztypen, die in der DB gespeichert sind. Der jeweilige Anwender braucht davon nur einen Teil, ein vom Schema abgeleitetes Subschema (Sicht, **view**).

Bsp.:

1. Anwendungsprogrammierer A hat die Aufgabe "Maschinelle Abwicklung des Bestellwesens"
A braucht dazu Sicht:

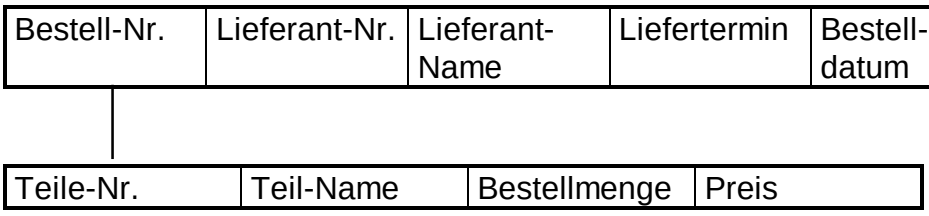


Abb. 1.3-33: Sicht des Anwendungsprogrammierers A

2. Anwendungsprogrammierer B ist für die maschinelle Abwicklung der Nachbestellungen aus dem Lagerwesen zuständig

B braucht dazu die Sicht:

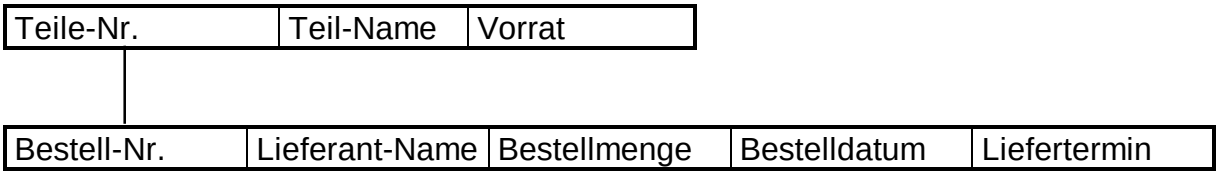


Abb. 1.3-34: Sicht des Anwendungsprogrammiereres B

Das konzeptuelle Schema soll eine vollständige redundanzfreie und neutrale Darstellung der logischen Datenstrukturen gewährleisten. Bei vollständiger Daten-neutralität ist es möglich durch unterschiedliche Benutzersichten (Netze, Hierarchien, Relationen) das Basisschema zu betrachten. Eine Sicht wird dabei mit Hilfe einer Datenmanipulationssprache aus der Basis erzeugt.
Auf der anderen Seite soll eine Reihe von Abbildungen verschiedene "Interne Schemata" zur Beschreibung der physischen Strukturen erzeugen. Sie enthalten mit ihren Katalogen die Details zu Zugriffspfaden und zur physischen Speicherung. Eine allgemeine Beschreibung der Schnittstellen enthält der ANSI/SPARC-Architekturvorschlag [25].

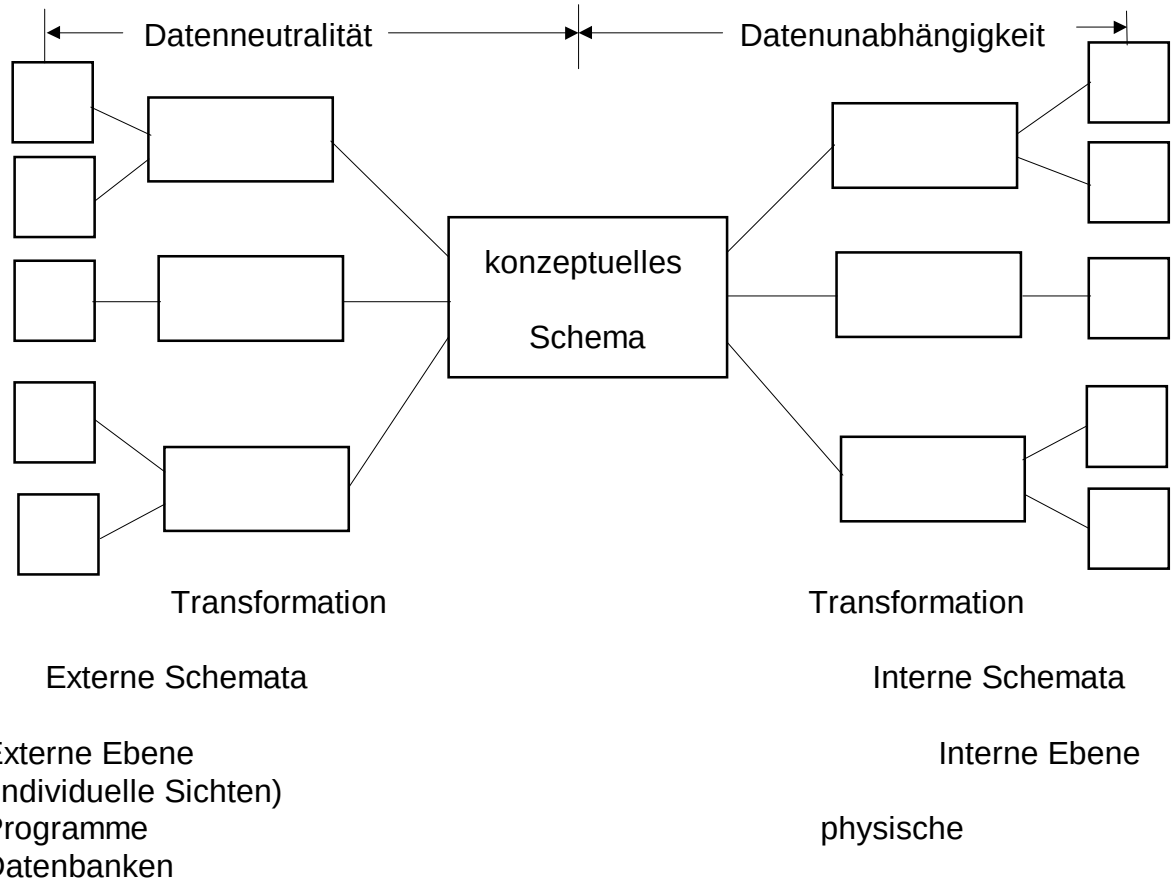


Abb. 1-3-35: Datenbankarchitektur nach ANSI/SPARC

[25] ANSI/X3/SPARC Group on Data Base Management Systems: Interim Report. Bulletin of ACM SIGMOD, No. 2, (1975)
ANSI(American National Standard Institute)/Sparc(Standard Planning and Requirement Committee

Die **ANSI/SPARC**-Architektur sieht 3 Ebenen vor:

1. Eine mittlere **konzeptuelle Ebene**, die alle drei Anwendersichten zu einer Art gemeinschaftlichen Sicht vereinigt
2. Eine **interne Ebene**, die es gestattet, unter Kenntnis von Anwenderprofilen und verfügbarer Hard- und Grundsoftware die leistungsfähigsten Speicher- und Zugriffsmethoden zu wählen
3. Eine **externe Ebene** mit beliebig vielen Anwendersichten

Im Mittelpunkt der verschiedenen Betrachtungen (Sichten) steht das allumfassende Konzept für die Anwendung der Daten. Datenbanken stellen auf der einen Seite die Mittel zur Verfügung, die die jeweiligen Dateien für die spezifischen Anwendungen bereitstellen. Auf der anderen Seite sorgen sie für die Speicherung der Daten (internes Schema, Speicherschema). Entscheidend ist in beiden Fällen das Datenmodell, das der allgemeinen Beschreibung zugrundeliegt. Das **Datenmodell** soll möglichst genau die realen Verhältnisse (die reale Welt) wiedergeben, d.h. die unterschiedlichen Objekte (Entitäten) und ihre Beziehungen. Ein **Datenmodell** ist dann das Muster (das Schema), nach dem die Daten logisch organisiert werden.
Im Hinblick zu den Anwenderprogrammen ist auf **Datenneutralität** und mit Blickrichtung auf die physische Datenbank auf **Datenunabhängigkeit** zu achten.

Datenneutralität bedeutet:
Neue Anwendungen und neue Benutzersichten sollen keinen Einfluß auf existierende Anwendungen und Sichten ausüben.
Datenunabhängigkeit bedeutet:
Neue Geräte, verbesserte Speichertechnologien, veränderte Zugriffspfade sollen sich in Anwendungen nur durch Leistungsverbesserung, nicht durch Funktionsänderung bemerkbar machen.

Bei vollständiger Datenneutralität ist es möglich, durch unterschiedliche Benutzersichten (Netze, Hierarchien, Relationen) das Basis-Schema zu betrachten. Eine Sicht (**view**) wird mit Hilfe der Datenmanipulationssprache (**DML**) aus der Basis, deren Struktur durch eine Datendefinitionssprache (**DDL**) festgelegt wurde, bestimmt.
Auf der anderen Seite sollte eine Reihe von Abbildungen das "Interne Schema" zur Beschreibung physischer Strukturen erzeugen. Sie enthalten mit ihren Katalogen die Details zu Zugriffspfaden und zur physischen Speicherung. Eine Speicherstrukturierungssprache (**SSL**) unterstützt die verschiedenen Alternativen zur physischen Speicherung.

Ein hohes Maß an Datenunabhängigkeit und -neutralität ist mit hierarchischen und auch mit netzwerkorientierten Datenbanken nicht zu erreichen.

Bsp.: Gegeben ist

RECHNUNG

RECH-NR.: 2010 DATUM: 31.12.1995
KUNDEN-NR.: 50118 NAME: ..... ADRESSE: .....

ARTIKEL-NR.	BEZEICHNUNG	PREIS	MENGE	BETRAG
A	Widerstand	5,00	2000	10.000,00
B	Spule	3,00	1000	3.000,00
....	.....	.....	.....	
			SUMME:	18.000,00

Abb. 1.3-36: Ein Rechnungsformular

Das zugehörige Datenbankstrukturdiagramm einer netzwerkorientierten Datenbank, das die Daten des vorliegenden Rechunungsformulars benutzt, sieht so aus:

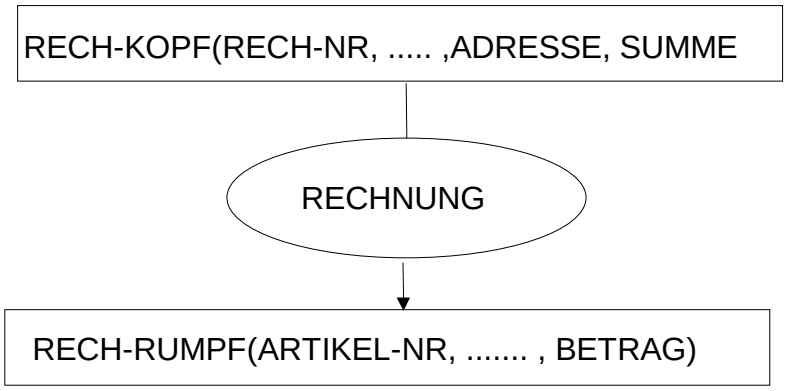


Abb. 1.3-37: Datenbankstrukturdiagramm zum Rechnungsformular

Besitzt dieser Entwurf für eine netzwerkorientierte Datenbank die Eigenschaften **Datenunabhängigkeit** und **Datenneutralität**?

Datenunabhängigeit?
Ein Zugriffspfad (, wesentlich bestimmt durch die Zeilenposition) ist zur Daten-wiedergewinnung unerläßlich.
Datenneutralität?
Es werden nur solche Anwendungen unterstützt, die zuerst den Rechnungskopf und dann den Rechnungsrumpf lesen.

Datenunabhängigkeit und Datenneutralität sind über ein netzwerkorientiertes DB-Modell nicht erreichbar[26].
Sind solche Eigenschaften mit dem relationalen DB-Modell möglich?
Da das Rechnungsformular die Struktur einer komplexen (geschachtelten) Tabelle aufweist, das relationale DB-Modell nur einfache Tabellen kennt, ist hier ein Nor-malisierungsprozeß durchzuführen. Normalisieren bedeutet: Darstellung des logischen Schemas einer relationalen Datenbank in Form einfacher, geschachtelter Tabellen. Aus dem vorliegenden, hierarchischen Schema werden 2 Relationen:

RECH-KOPF(RECH-NR,KUNDEN-NR,DATUM,NAME,ADRESSE,SUMME)
RECH-RUMPF(RECH-NR,ARTIKEL-NR,BEZ,PREIS,MENGE,BETRAG)

Das Schema für die relationale Datenbank zeigt die gewünschte **Unabhängigkeit**:
Eine Rumpfzeile kann angesprochen werden, ohne zum Kopf zugreifen zu müssen. Voraussetzung dafür ist: Es existiert ein Primärschlüssel im Rechnungskopf (RECH-NR).

1.3.7 Das Datenbankmodell für objektorientierte Datenbanken

Grundlagen

Ein **objektorientiertes Datenbanksystem** ist auch nichts Anderes als ein Datenbanksystem mit den üblichen Leistungen (Datenintegration, Daten-unabhängigkeit, Unterstützung vom Mehrbenutzerbetrieb, Gewährleistung der Datensicherheit, Datenschutz, Datenkonsistenz, Verarbeitungsintegrität). Allerdings stützt sich ein derartiges Datenbanksystem auf ein eigenes Datenbankmodell (objektorientiert, `OODM`) ab und darin liegt die Problematik: Eine einheitliche, allgemeingültige Definition für ein **OODM** gibt es noch nicht.
Ausgangspunkt ist das Ziel: Umweltsachverhalte beliebiger Art und Komplexität möglichst genau (`1:1` Abbildung) beim Datenbankentwurf wiederzugeben. Ein Umwelt-Objekt soll einem Datenbankobjekt entsprechen.
Auch komplex aufgebaute Objekte sollen modelliert werden können, denn Objekte können Bestandteile besitzen, die wiederum in Unterobjekten beschrieben werden können.
Im Schema werden, wie üblich Objekttypen spezifiziert. Zusätzlich zur Festlegung der gewöhnlichen Attribute (sog. einfache Datentypen: `integer`, `real`, `string`, `date`, `time`) und deren Wertebereiche ist hier noch anzugeben, von welchem Typ evtl. Unterobjekte sein können. Für den Aufbau komplexer Strukturen sollen außerdem nicht nur Konstruktoren für Tupel, sondern auch für Mengen[27] und Listen zur Verfügung stehen.
Zum Konzept für komplexe Objekte zählen auch entsprechende Operatoren. Sie dienen zum Auf- und Abbau (Konstruktoren und Destruktoren) und zum Zugriff auf

[26] vgl.: Wedekind, Hartmut: "Relationale Datenbanksysteme", Informatik-Spektrum, Nr.5, !978, Seiten 5 - 16
[27] z.B. ein Typkonstruktor zur Mengenbildung mit SET OF

die Eigenschaften von Objekten bzw. zur Bildung neuer Objekte aus existierenden Objekten (sog. Umsetzungsoperatoren). Objekte können nur über den Aufruf der definierten Operatoren (Methoden) angesprochen bzw. bearbeitet werden (**Konzept der Datenkapselung**). Lediglich über die Implementierung der Operatorprogramme können direkte Zugriffe auf die Wertepräsentation codiert werden.
Die Definition neuer Objekttypen schließt die für diese Typen charakterischen Operatoren (**Methoden**) mit ein. Objekktypen werden in diesem Zusammenhang **Klassen** genannt.
Objektorientierte Systeme unterstützen eine Reihe von Klassen (sog. Basisklassen), die häufig verwendete Datenstrukturen (Listen, Mengen, etc.) bereithalten. Basisklassen sind evtl. in Klassenbibliotheken verfügbar.
In einem OODM müssen alle Objekte eigenständig, von ihren aktuelle Werten unabhängig, identifiziert werden und zu diesem Zweck eine eindeutige, unveränderliche Kennung erhalten (**Objektidentifikator**, **Surrogat**). Dadurch ist der Anwender entlastet von der Festlegung eindeutiger, unveränderlicher Schlüssel für alle Objekte (, er darf dies auf Wunsch aber auch weiterhin tun). Jedes Objekt ist über seinen Identifikator ansprechbar. Gleichheit und Identität von Objekten können unterschieden werden.
Innerhalb des objektorientierten Datenbanksystems (OODBS) erfolgen alle Objektidentifizierungen über Surrogate. Das sind systemgenerierte, global eindeutige Bezeichner, die unabhängig vom physischen Speicherungsort sind (Ablageunabhängigkeit). Der Benutzer hat keinen Einfluß auf Surrogate. Benutzerschlüssel sind deshalb nicht überflüssig und können weiterhin zur Identifizierung von Objekten auf Benutzerebene interessant sein. Benutzerdefinierte Schlüssel zählen in objektorientierten Systemen zu den Eigenschaften des Objekts und haben nicht die Bedeutung wie in relationalen Systemen.
In einem OODM kann ein Typ Untertyp eines anderen sein. Der Untertyp ist eine **Spezialisierung** des Obertyps bzw. der Obertyp ist eine **Generalisierung** des Untertyps. Man spricht von der „IS_A“-Beziehung zwischen Unter- und Obertyp. Für den Untertyp müssen lediglich die gewünschten zusätzlichen Eigenschaften festgelegt sein, seine Objekte erben automatisch alle für den Obertyp definierten Eigenschaften.

Entwicklungslinien objektorientierter Datenbanksysteme

Man kann zwei Entwicklungslinien bei objektorientierten Datenbanksystemen (OODBS) [28] erkennen:

- Erweiterung bzw. Ergänzung objektorientierter Programmiersprachen (OOPL)
  Ausgangspunkt ist eine **OOPL** (z.B. C++, Smalltalk). Durch Ergänzung mit Datenbankkonzepten (Dauerhaftigkeit (Persistenz), Speicherstrukturen, Zugriffspfade, Transaktionen, Concurrency Control, Recovery-Mechanismen) wird ein OODBS entwickelt. Außerdem werden vordefinierte Klassen und Methoden mitgeliefert, die fehlenden Typkonstruktoren nachahmen (etwa SET OF) und Abfrageoperatoren bereitstellen (etwa das Auswählen von Objekten aus einer Menge nach

[28] vgl. Dittrich, Klaus R.: Objektorientiert, aktiv, erweiterbar: Stand und Tendenzen der "nachrelationalen" Datenbanktechnologie, it 5/90, Seiten 343 - 353

bestimmten Bedingungen). Beispiele sind Datenbanken, die entweder auf Smalltalk[29]- oder C++[30]-Basis implementiert werden.
- Überlagerung relationaler Systeme mit Objektstrukturen
  Die bewährte relationale Datenbank-Technologie wird beibehalten und graduell um objektorientierte Eigenschaften ergänzt.

Es gibt auch Neuentwicklungen voll objektorientierter Datenbankmodelle. Sie sind vollständig auf Erfordernisse der Datenbank und auf Objektorientierung zugeschnitten. Da genaue Kriterien für Konzepte objektorientierter Datenbanken erst seit 1989[31] vorliegen, sind Implementierungen noch nicht sehr zahlreich.

„Object Oriented Database Manifesto“

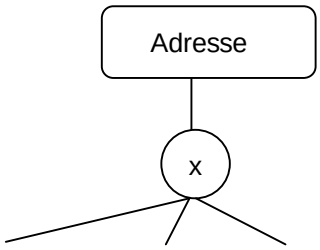
In diesem Bericht[32] haben Datenbankforscher Richtlinien für das objektorientierte Datenbankmodell zusammengefaßt. Merkmale von objektorientierten Datenbanksysteme wurden hier in drei Kategorien eingeteilt: Pflchtmerkmale, optionale Merkmale, offene Merkmale.
Zu den aus der objektorientierten Programmierung entlehnten Pflichtmerkmalen zählen:

1. Objektbegriff / Objektidentität
In objektorientierten Datenbankmodellen besteht ein Objekt aus

- einem Objektidentifikator
- einer Menge von Werten (Attribute, Variable)
- einer Menge von Prozeduren / Operatoren[33]

Der Objektidentifikator wird vom DBMS vorgegeben[34] und werwaltet. Er ist systemweit eindeutig und während der Objektlebensdauer konstant.

2. Komplexe Objekte[35]
Ausgangspunkt sind elementare Werte, z.B. Zeichenketten oder Zahlen, aus denen mit Hilfe sog. Konstruktoren komplexe Werte gebidet werden. Einfache Konstruktoren sind:
- Tupelkonstruktor (Aggregation): Daten setzen sich häufig aus anderen Daten zusammen, auch die repräsentativen (physischen oder logischen) Gegenstände der relen Welt sind häufig aus anderen Gegenständen zusammengesetzt, z.B. das Aggregat `Adresse`:



[29] Gemstone
[30] Object-Store, ONTOS, POET
[31] Atkison et al.: Object Oriented Database System Manifest
[32] Atkison, M.: The object oriented database manifesto, Proceedings First national Conference on Deductive and Object Oriented Databases, Kyoto 1989
[33] Die 3 Begriffe werden synonym verwendet
[34] Er beruht nicht auf den aktuellen Werten des Objekts (wie etwa der Primärschlüssel bei relationalen datenbanken)
[35] Synonym verwendete Begriffe: Strukturierte Objekte, Zusammengesetzte Objekte

Abb.: Aggregat „Adresse“ in erweiterter ERM-Darstellungf

Häufig wird der Tupelkonstruktor mit „TUPLE OF“ notiert.

- Mengenkonstruktor (Gruppierung): Dadurch wird ein neuer Typ erzeugt, der aus mehreren Instanzen des in die Gruppierung eingehenden Typs eine Instanz des neuen Typs formt. Häufig wird der Mengenkonstruktor mit „SET OF“ notiert.

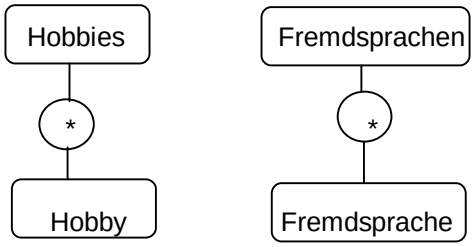


Abb.: „Gruppierungen in erweiterter ERM-Diagrammdarstellung

- Listenkonstruktor: „geordnete Menge“
- Feldkonstruktor: „`array`“, in dem Elemente über einen Index (Subskript) ansprechbar sind.

3. Klassen, Typen

Eine Klasse ist eine Beschreibung, die
- einen Namen für die Objektmenge
- die Struktur der Objekte der Menge (ihre Werte)
- die Methoden, die die Objekte der Menge ausführen können

umfaßt. Ein Objekt einer Klasse wird Instanz genannt. Instanzen einer Klasse haben den gleichen Aufbau, benutzen diesselben Namen, Typen und Methoden. Aus der Klasse können Instanzen ins Leben gerufen werden.

4. Typhierarchie, Vererbung

Es können übergeordnete Klassen an untergeordnete Klassen Eigenschaften vererben. Die Eigenschaften können Zustand und / oder das Verhalten beschreiben.

5. Einkapselung

Zustand und Methoden sind für den Anwender gekapselt, d.h. für ihn unsichtbar (information hiding). Er kann nicht zu den Daten direkt zugreifen, sondern nur über die vorhandenen Methoden. Die Methoden sind ihm über Schnittstellen bekannt, über die er mit geeigneten Nachrichten die Methoden zum Arbeiten veranlassen kann.

6. Berechnungsvollständigkeit

Zur Realisierung der Methoden muß eine Sprache zur Verfügung stehen, die die Formulierung beliebiger Algorithmen gestattet. Die Datenbanksprache SQL in aktuell vorliegender Form genügt diesen Anforderungen nicht.

7. Überladen / Überschreinen und spätes Binden

Dabei handelt es sich um Detaikonzepte, die die vorteile der Klassenhierarchie und Vererbung zur Geltung bringen.
Überladen wird durch die gleiche Namensvergabe für verschiedene Operatoren erzwungen.
Überschreiben ist eine Methode, die sowohl in der übergeordneten Klasse als auch in der untergeordneten Klasse definiert ist. Für die Instanzen der untergeordneten Klasse ist die Methode der untergeordneten Klasse maßgebend.
Spätes Binden bedeutet: Das Binden eines Operatornamens an den zugehörigen ausführbaren Programmcode erfolgt dynamisch erst zur Laufzeit.

Standard für objektorientierte Datenbanken: ODMG-93

Die Mitgliederfirmen der ODMG (Object Database Management Group)[36] haben einen Standard für objektorientierte Datenbanksysteme (ODBS)[37] definiert mit Standardisierungsvorschlägen für die Bereiche

- Objektmodell
- Objekt-Definitionssprache (**ODL**)
- Objekt-Abfragesprache (**OQL**)
- C++- und Smalltalk-Anbindung für **ODL** und **OQL**, sowie **OML** (Object Manipulation Language)

Anbindung für Programmiersprachen

Eine ODMG-ODBS sieht dafür folgende Möglichkeiten vor

- Definition von Datenbankschemata
- Einfüllen von Daten
- Zugriff und Manipulation von Daten
- Verwendung der OQL

Das „Typ“-System der Programmiersprache und das Datenbankmodell des Datenbanksystems sollen möglichst integriert werden können. So soll das DBS mit dem erweiterten „Typ“-System der jeweiligen Programmiersprache definiert werden können.

[36] eine Gruppe führender Hersteller objektorientierter Datenbanksysteme
[37] Spezifikation ODMG-93

1.3.8 Die UML zur Beschreibung von Datenbankanwendungen

Die Unified Modelling Language (UML)[38] ist eine Sprache zur Beschreibung von Softwaresystemen. Sie besteht aus verschiedenen Diagrammen, die wiederum verschiedene grafische Elemente besitzen. Die Bedeutung (, also die Semantik,) der Elemente ist genau festgelegt. Innerhalb der UML gibt es allerdings für ein und denselben Sachverhalt mehrere Darstellungsarten.

1.3.8.1 Die Diagramme

Die Notation der UML umfaßt Diagramme für die Darstellung verschiedener Ansichten auf das System. Sie sind vergleichbar mit Baupänen.

Diagramm	Einsatzgebiet
Use-Case	Geschäftsprozesse, allgemeine Einsatzmöglichkeiten
Klassendiagramm	So gut wie überall, das Klassendiagramm ist das wichtigste Diagramm der UML
Interaktionsdiagramm	Zeigt den Nachrichtenfluß und damit die Zusammenarbeit der Objekte im zeitlichen Ablauf
- Sequenzdiagramm	Zeitliche Aufrufstruktur mit wenigen Klasse
- Kolloborationsdiagramm	Zeitliche Aufrufstruktur mit wenigen Nachrichten
Package-Diagramm	Groborientierung, in welchem Modul welche Klasse zu finden ist. Aufteilung in Unterprojekte, Bibliotheken, Übersetzungseinheiten.
Zustandsdiagramm	Darstellung des dynamischen Verhaltens
Aktivitätsdiagramm	Bei parallelen Prozessen und anderer Parallelität, Geschäftsprozesse
Implementierungsdiagramm	Besonders für die Darstellung von verteilten Anwendungen und Komponenten; allgemein Darstellung von Implementierungs-einheiten (Übersetzungseinheiten, ausführbare Programme, Hardwarestruktur)
- Komponentendiagramm	Zusammenhänge der Software
- Deployment-Diagramm	Hardwareaufbau

Abb.: Einsatzgebiete und Eigenschaften der verschiedenen UML-Diagramme

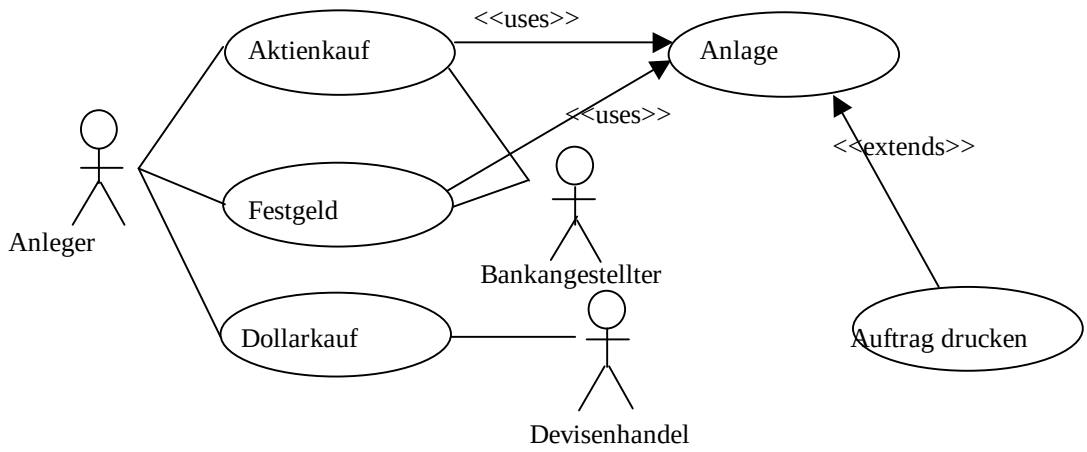
Jedes der Diagramme besitzt zahlreiche Stilelemente mit verschiedenen Beschriftungsarten[39].

Use Case

Anwendungdfälle kann man sich als eine Sammlung von Szenarios über dem Systemeinsatz vorstellen. Jedes Szenario beschreibt eine Sequenz von Schritten. Jede dieser Sequenzen wird von einem Menschen, einem anderen System, einem Teil der Hardware oder durch Zeitablauf in Gang gesetzt.
Entitäten, die solche Sequenzen anstoßen, nennt man Akteure. Das Ergebnis dieser Sequenz muß etwas sein, was entweder dem Akteur, der sie initiierte oder einem anderen Akteur nutzt.

[38] 1996 begannen die Arbeiten an der UML. Im Jahre 1997 wurde die Version 1.0 bei der Object Management Group (OMG) als Standardisierungsvorschlag eingereicht, und eine Beschreibung der Version 1.0 wurde veröffentlicht. Im September 1991 wurde die Version 1.1. bei der OMG eingereicht.
[39] Vgl.: Günther Wahl: UML kompakt, OBJEKTspektrum 2/1998

Ein „Use Case“ ist eine typische Handlung, die ein Benutzer mit dem System ausführt, z.B. Aktienkauf. Im Diagramm werden Use Cases durch Ellipsen dargestellt. Verbindungen zu den entsprechenden Use-Cases werden durch Linien dargestellt. Damit wird angezeigt, welche Aktoren an dem entsprechenden Use Case beteiligt sind.
Bsp.: „Geld anlegen“



In diesem Use-Case-Diagramm gibt es zwei weitere Verbindungstypen: <<uses>> und <<extends>>.
<<uses>> wird verwendet, wenn zwei oder mehrere Use-Cases einen ähnlichen Aufbau haben und Verhalten durch Kopieren wiederholt dargestellt werden muß. Mit <<uses>> ist es möglich, Verhalten von den Use-Cases in einen seperaten Use-Case zu verlagern.
Bei <<extends>> wird das Verhalten erweitert. So stellt „Auftrag drucken“ dem Use-Case Anlage zusätzliche Funktionalität zur Verfügung

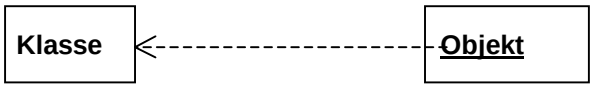
Abb.: Use-Case „Geld anlegen“

<<uses>> bzw. <<extends>> sind Darstellungen spezieller Stereotypen. Stereotypen sind Erweiterungsmechanismen der UML. Sie haben etwa die Bedeutung „so ähnlich etwa wie“und erlauben eine weitere Einteilung der Klassen in Schnittstellen-, Kontroll- und Entitätenobjekte (irgendwelche Dinge).

Elemente und Darstellung des Klassendiagramms

Das Klassendiagramm beschreibt die statische Struktur der Objekte in einem System sowie ihre Beziehungen untereinander. Die **Klasse** ist das zentrale Element. Klassen werden durch Rechtecke dargestellt, die entweder den Namen der Klasse tragen oder zusätzlich auch Attribute und Operationen. Klassenname, Attribute, Operationen (Methoden) sind jeweils durch eine horizontale Linie getrennt. Klassennamen beginnen mit Großbuchstaben und sind Substantive im Singular.
Ein strenge visuelle Unterscheidung zwischen Klassen und Objekten entfällt in der UML. Objekte werden von den Klassen dadurch unterschieden, daß ihre Bezeichnung unterstrichen ist. Auch können Klassen und Objekte zusammen im Klassendiagramm auftreten.

Wenn man die Objekt-Klassen-Beziehung (Exemplarbeziehung, Instanzbeziehung) darstellen möchte, wird zwischen einem Objekt und seiner Klasse ein gestrichelter Pfeil in Richtung Klasse gezeichnet:



Die Definition einer Klasse umfaßt die „bedeutsamen“ Eigenschaften. Das sind:

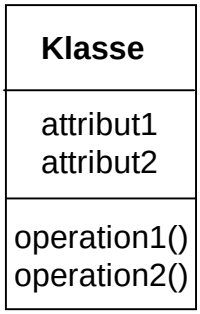
- Attribute
  d.h.: die Struktur der Objekte: ihre Bestandteile und die in ihnen enthaltenen Informationen und Daten.. Abhängig von der Detaillierung im Diagramm kann die Notation für ein Attribut den Attributnamen, den Typ und den voreingestellten Wert zeigen:
  Sichtbarkeit Name: Typ = voreingestellter Wert
- Operationen
  d.h.: das Verhalten der Objekte. Manchmal wird auch von Services oder Methoden gesprochen. Das Verhalten eines Objekts wird beschrieben durch die möglichen Nachrichten, die es verstehen kann. Zu jeder Nachricht benötigt das Objekt entsprechende Operationen. Die UML-Syntax für Operationen ist:

  Sichtbarkeit Name (Parameterliste) : Rückgabetypausdruck (Eigenschaften)

  Sichtbarkeit ist + (öffentlich), # (geschützt) oder – (privat)
  Name ist eine Zeichenkette
  Parameterliste enthält optional Argumente, deren Syntax dieselbe wie für Attribute ist
  Rückgabetypausdruck ist eine optionale, sprachabhängige Spezifikation
  Eigenschaften zeigt Eigenschaftswerte (über String) an, die für die Operation Anwendung finden
- Zusicherungen
  Die Bedingungen, Voraussetzungen und Regeln, die die Objekte erfüllen müssen, werden Zusicherungen genannt. UML definiert keine strikte Syntax für die Beschreibung von Bedingungen. Sie müssen nur in geschweifte Klammern ({}) gesetzt werden.
  Idealerweise sollten regeln als Zusicherungen (engl. assertions) in der Programmiersprache implementiert werden können.
  Bsp.: Eigenschaften eines Kreises
  Zu den Eigenschaften eines Kreises, der auf einem Bildschirm dargestellt werden soll, gehören:

  - Attribute: Radius des Kreises, Position des Kreises auf dem Bildschirm
    - Operationen: Anzeigen und Entfernen des Kreises, Verschieben des Kreises, Verändern des Radius des Kreises
  - Zusicherungen: Der Radius darf nicht negativ sein und nicht Null sein (radius>0)

Attribute werden mindestens mit ihrem Namen aufgeführt und können zusätzliche Angaben zu ihrem Typ (d.h. ihrer Klasse), einen Initialwert und evtl. Eigenschaftswerte und Zusicherungen enthalten. Attribute bilden den Datenbestand einer Klasse.
Operationen (Methoden) werden mindestens mit ihrem Namen, zusätzlich durch ihre möglichen Parameter, deren Klasse und Initialwerte sowie evtl. Eigenschaftswerte und Zusicherungen notiert. Methoden sind die aus anderen Sprachen bekannten Funktionen.



Bsp.: „Automatische Geldausgabe“

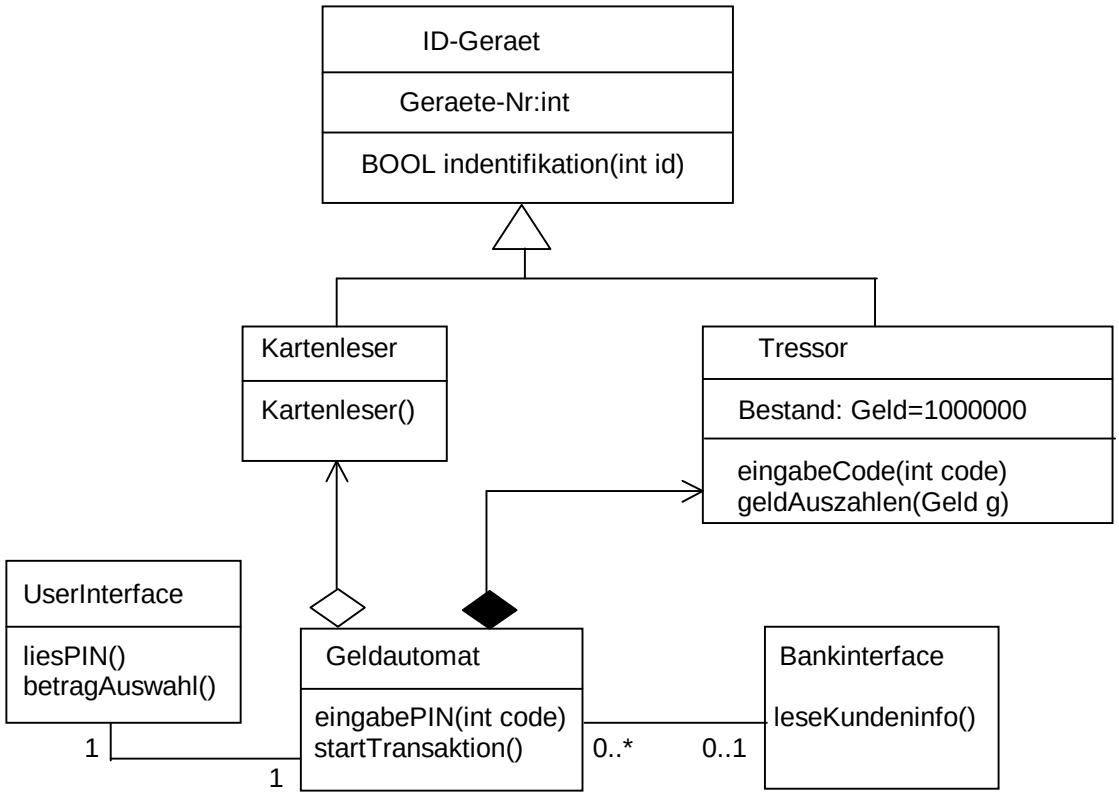
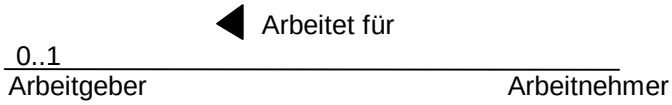


Abb.: Klassendiagramm „automatische Geldausgabe“

Die Klasse ist das zentrale Element, sie wird als Rechteck dargestellt (z.B. Geldautomat“). Die gefundenen Klassen sind durch Linien miteinander verbunden. Diese Linien stellen die Elemente Assoziation, Aggregation, Komposition und Vererbung dar.

Eine Assoziation (association) ist eine Strukturbeziehung, die eine Menge von Objektbeziehungen beschreibt. Eine Aggregation ist eine Sonderform der Assoziation. Sie repräsentiert eine (strukturelle) Ganzes/Teil-Beziehung. Grafisch wird eine Assoziation als durchgezogene Line wiedergegeben, die gerichtet sein kann, manchmal eine Beschriftung besitzt und oft noch weitere Details wie z.B. Muliplizität (Kardinalität) oder Rollenanmen enthält, z.B.:



Eine Assoziation kann einen Namen zur Beschreibung der Natur der Beziehung („Arbeitet für“) besitzen. Damit die Bedeutung unzweideutig ist, kann man dem

Namen eine Richtung zuweisen: Ein Dreieck zeigt in die Richtung, in der der Name gelesen werden soll.
Rollen („Arbeitgeber, Arbeitnehmer) sind Namen für Klassen in einer Relation. Eine Rolle ist die Seite, die die Klasse an einem Ende der Assoziation der Klasse am anderen Ende der Assoziation zukehrt. Die Navigierbarkeit kann durch einen Pfeil in Richtung einer Rolle angezeigt werden.

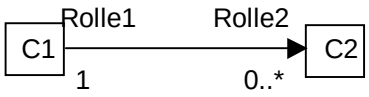


Abb.: Binäre Relation R = C1 x C2

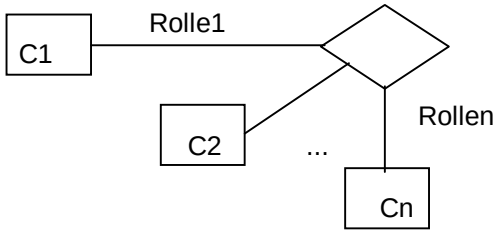


Abb.: n-äre Relation C1 x C2 x ... x Cn

In vielen Situationen ist es wichtig anzugeben, wie viele Objekte in der Instanz einer Assoziation miteinander zusammenhänen können. Die Frage „Wie viele?“ bezeichnet man als Multiplizität der Rolle einer Assoziation. Gibt man an einem Ende der Assoziation eine Multiplizität an, dann spezifiziert man dadurch: Für jedes Objekt am entgegengesetzten Ende der Assoziation muß die angegebene Anzahl von Objekten vorhanden sein.

1:1	1..*
1:1..n	0..* 2..6
0..n:2..6	0..* *
0..n:0..n	17 4
17:4	n m
?	

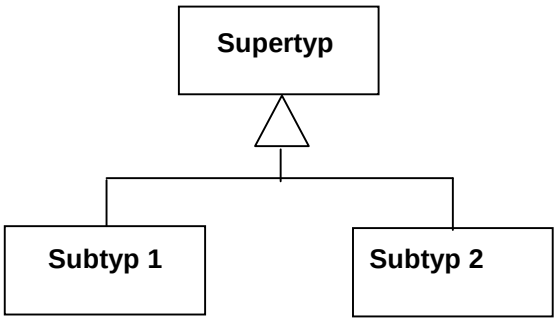
Abb.: Kardinalitäten für Beziehungen

Jede Assoziation[40] kann eine Richtung besitzen. Diese bestimmt ein Pfeil am Ende der Assoziation. Zugriffe können dann nur in Pfeilrichtung (Navigationsfähigkeit) erfolgen.
Eine Aggregation wird durch eine Raute dargestellt, z.B. zwischen „Kartenleser“ und „Geldautomat“[41]. Sie gibt an:

Die Klasse „Kartenleser“ ist in der Klasse „Geldautomat“ enthalten (**Ist-Teil-von-Beziehung**)

[40] Eine Navigation mit Pfeil kann als Zeiger in einer Programmiersprache betrachtet werden.
[41] vgl. Abb.: Klassendiagramm „automatische Geldausgabe“.

Die Komposition wird durch eine ausgefüllte Raute dargestellt und beschreibt ein „physikalisches Enthaltensein“.
Die **Vererbung** (Spezialisierung bzw. Generalisierung) stellt eine Verallgemeinerung von Eigenschaften dar. Eine Generalisierung (generalization) ist eine Beziehung zwischen dem Allgemeinen und dem Speziellen, in der Objekte des speziellen Typs (der Subklasse) durch Elemente des allgemeinen Typs (der Oberklassse) ersetzt werden können. Grafisch wird eine Generalisierung als durchgezogene Linle mit einer unausgefüllten, auf die Oberklasse zeigenden Pfeilspitze wiedergegeben, z.B.:



Interaktionsdiagramm

Es gibt zwei Arten von Interaktionsdiagrammen:

- Sequenzdiagramme
- Kollabarationsdiagramme

Die beiden Diagramme beschreiben zeitliche Abläufe, d.h. Aufrufsequenzen.
Ausgangspunkt von Interaktionsdiagrammen sind Fallbeispiele (Szenarios). Beim Erstellen der Diagramme konzentriert man sich auf die wichtigsten Fälle. Anschließend werden die Sonderfälle mit einbezogen.

1. Sequenzdiagramme

Ein Sequenzdiagramm ist ein Interaktionsdiagramm, bei dem die zeitliche Abfolgr der Nachrichten im Vordergrund steht. Sequenzdiagramme visualisieren

- die Lebensdauer von Objekten (gestrichelte Linie, die die Existenz eines Objekts während eines Zeitraums darstellt). Die Zeitachse verläuft von oben nach unten. Objekte sind als Rechtecke am oberen Rand von gestrichelten (Lebens-) Linien dargestellt. Am linken Rand können noch Kommentare stehen.
- die Aktivität von Objekten. Der Focus-of-Control (das langezogene Rechteck) gibt an, wo eine Aktivität ausgeführt wird. Ist ein Objekt ohne Aktivität vorhanden, wird dies durch eine gestrichelte Linie angezeigt.Objekte werden durch einen Pfeil auf das Rechteck erzeugt, ein Löschen wird durch ein Kreuz dargestellt.
- Nachrichenaustauch zwischen Objekten (beschriftete Pfeile). Über den Pfeilen stehen Operationen, die ausgeführt werden. In eckigen Klammern können Bedingungen angegeben werden, wann die Operation ausgeführt wird. Ruft das Objekt eine Operation von sich selbst auf, dann zeigt der Pfeil auf die Lebenslinie zurück.

Die in den Anwendungsfällen beschriebenen Objektinteraktionen können in Sequenzdiagrammen verfeinert werden. Für jeden Anwendungsfall können ein oder mehrere Sequenzdiagramm(e) konstruiert werden. Sequenzdiagramme stammen aus der Telekommunikationstechnik (z.B. für Protokolle).

Bsp.: Ablauf des „Use Cases“: „Neue Vorlesung anbieten“

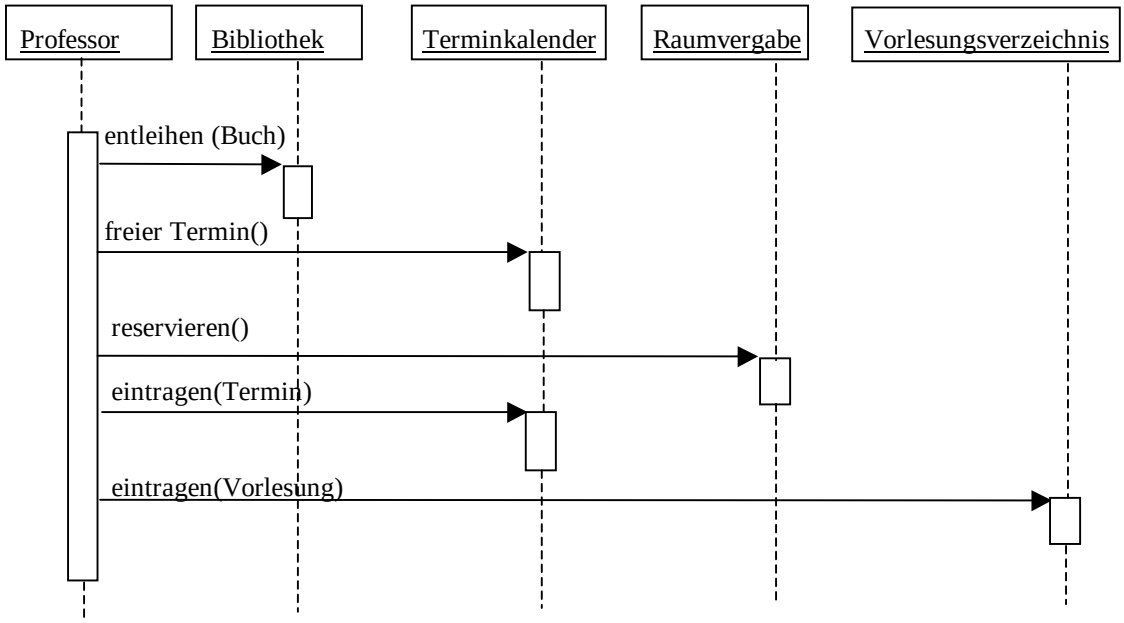


Abb.: Sequenzdiagramm: „Neue Vorlesung anbieten“

2. Kollabarationsdiagramme

Ein Kollaborationsdiagramm ist ein Interaktionsdiagramm, in dem die strukturelle Organisation der Objekte im Vordergrund steht, die Nachrichten senden bzw. empfangen.Sequenz- und Kollaborationsdiagramme sind isomorph, so daß man das eine in das andere umwandeln kann. Sequenzdiagramme haben zwei Merkmale, die sie von Kolloborationsdiagrammen unterscheiden:
1. die Objektlebenslinie (senkrechte gestrichelte Linie)
2. Kontrollfokus (langes, schmales Rechteck)

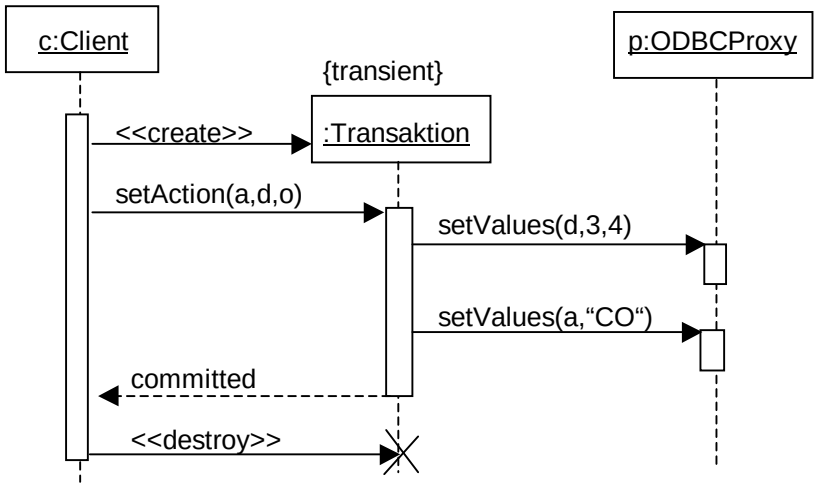
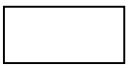


Abb.: Sequenzdiagramm

Kollaborationsdiagramme haben zwei Merkmale, die sie von Sequenzdiagrammen unterscheiden:

1. Zur Anzeige, wie ein Objekt mit einem anderen Objekt verbunden ist, kann man einen Pfad Stereotyp am entferntesten Ende der Objektbeziehung angeben (wie <<local>> zur Kennzeichnung, dass dieses Objekt für das zu sendende Objekt lokal ist).
2. Zur Kennzeichnung der zeitlichen Anordnung schreibt man eine Nummer als Präfix vor die Nachricht (beginnend mit der Nummer 1, monoton wachsend für jede weitere Nachricht für den Kontrollfluss).



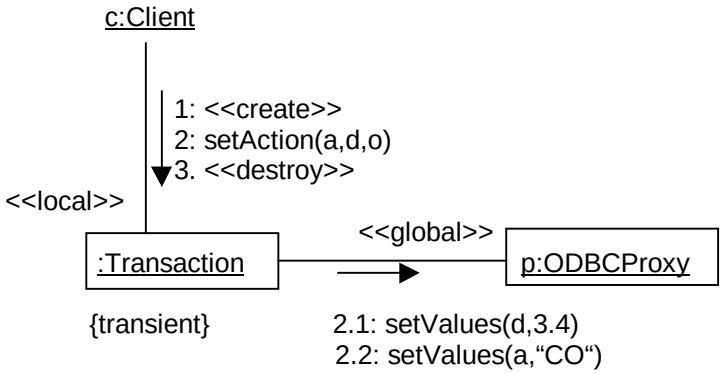


Abb.: Kolloborationsdiagramm

Zustandsdiagramm

Ein Zustandsdiagramm zeigt einen Automaten, der aus Zuständen, Zustandsübergängen, Ereignissen und Aktivitäten besteht. Bei Zustandsdiagrammen steht das nach Ereignissen geordnete Verhalten eines Objekts im Vordergrund.
Zustände werden symbolisch über Rechtecke mit abgerundeten Ecken dargestellt. Durch das Eintreffen von Ereignisssen kann ein anderer Zustand erreicht werden, was durch Pfeile angedeutet wird. Die Pfeile haben eine Beschriftung für das auslösende Ereignis und stellen Übergänge dar. In den Zuständen werden Operationen mit einer gewissen Zeitdauer ausgeführt, die Aktivität genannt werden. Im Gegensatz dazu wird die Zeitdauer von Aktionen mit Null angenommen. Aktionen sind Operationen, die an Übergängen ausgeführt werden. Angezeigt werden Aktionen durch einen Schrägstrich vor dem Namen. Selbstverständlich können Übergänge auch an Bedingungen, die in eckigen Klammern stehen, geknüpft werden.

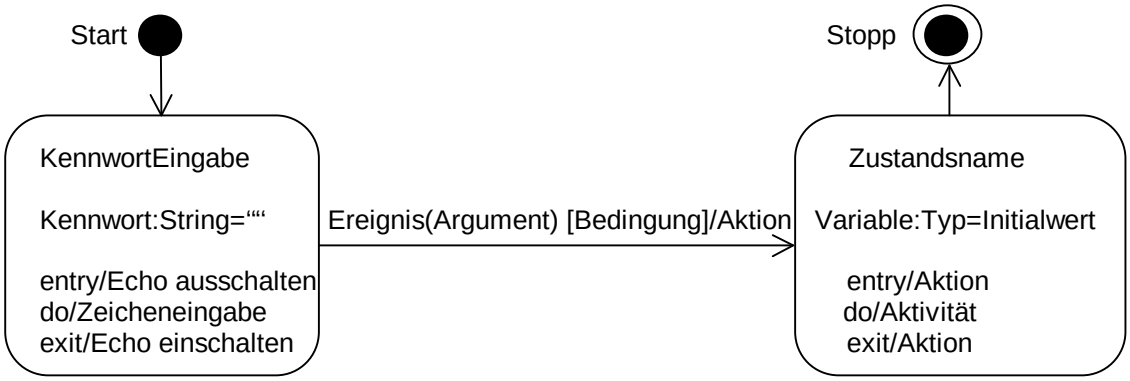
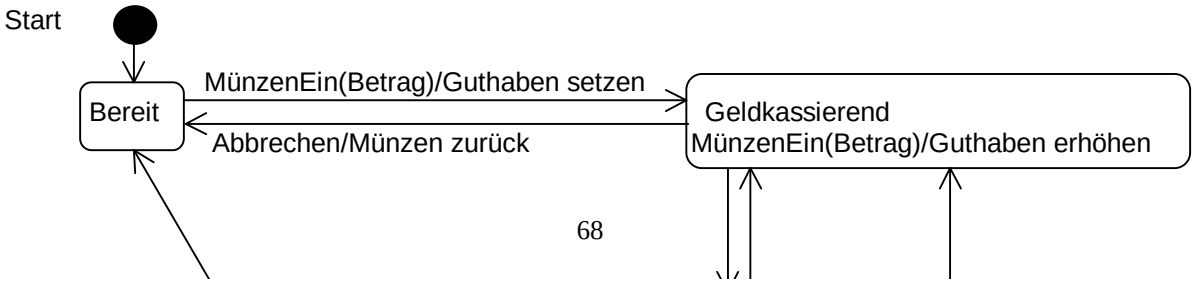


Abb.: Symbole im Zustandsdiagramm

Die Wörter `entry`, do, `exit` sind reserviert und können als Bezeichnungen für Ereignisse nicht verwendet werden.

`entry` gibt an, welche Aktion im Hintergrundausgeführt wird.
`exit` gibt an, welche Aktion beim Verlassen des Zustands ausgeführt wird
do beschreibt den Aufruf eines verschaltelten Zustandsdiagramms. Allgemein bezeichnet do eine Aktivität, die (unendlich) lange dauern kann.



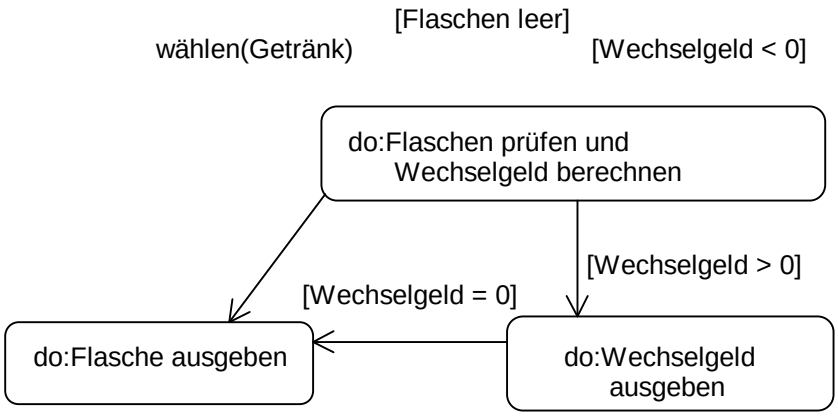


Abb.: Zustandsdiagramm für einen Getränkeautomaten

Aktivitätsdiagramm

Ein Aktivitätsdiagramm ist ein Sonderfall des Zustandsdiagramms, der den Fluß von einer Aktivität zu einer anderen innerhalb eines Systems zeigt. Eine Aktivität (actvity) ist ein andauernder, nichtatomarer Ablauf innerhalb eines Automaten. Aktivitätsdiagramme enthalten üblicherweise

- Aktivitätszustände und Aktionszustände
- Zustansübergänge (Transitionen)
- Objekte

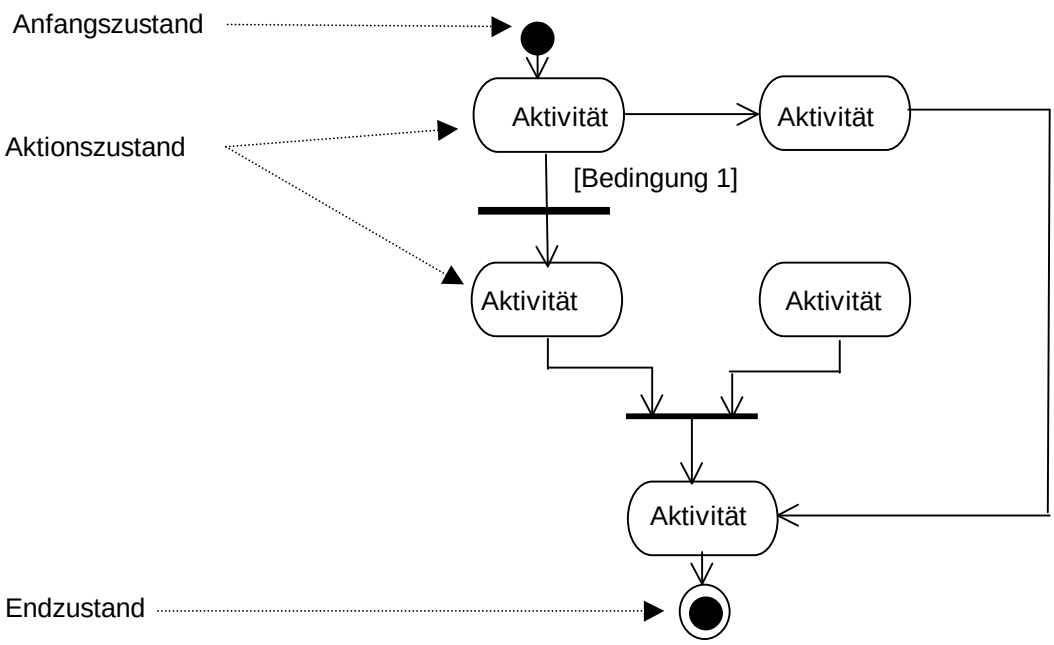


Abb.: Aktivitätsdiagramm

Package-Diagramm

Package-Diagramme dienen zur Strukturierung der verschiedenen Darstellungen. Damit werden Gruppen von Diagrammen oder Elementen zusammengefaßt.

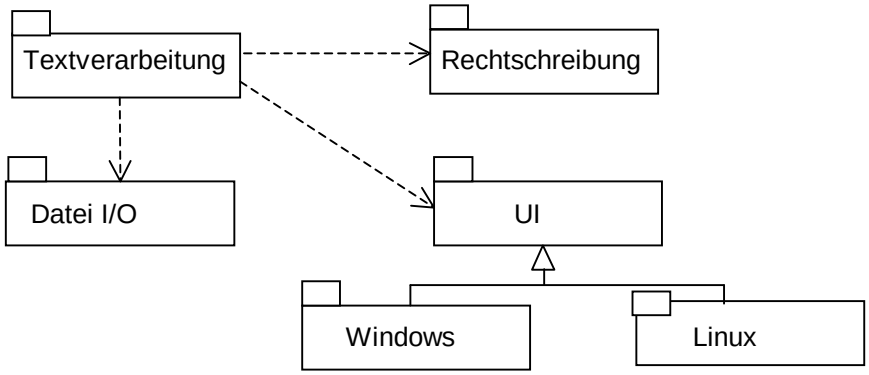


Abb.: Beispiel für ein Package-Diagramm

Ein Package-Diagramm besteht im wesentlichen aus Packages (dargestellt durch große Rechtecke mit kleinem Rechteck links oben) und Abhängigkeiten (den gestrichelten Pfeilen). Eine Abhängigkeit gibt an: Bei einer Änderung des Packages an der Pfeilspitze muß das Package am anderen Ende der gestrichelten Linie evtl. geändert und neu übersetz werden. Packages können weitere Packages enthalten.

Implementierungsdiagramm

Implementierungsdiagramme zeigen Aspekte der Implementierung. Diese umfassen die Codestruktur und die Struktur des Systems zur Ausführungszeit. Es gibt zwei Formen

Komponentendiagramm
Einsatzdiagramm (Deploymentdiagramm)

1. Komponentendiagramm

Das Komponentendiagramm zeigt die Abhängigkeit unter den Softwarekomponenten, d.h.: die Abhängigkeiten zwischen Quellcode, Binärcodekomponenten und ausführbaren Programmen. Einige dieser Komponenten existieren nur während des Übersetzungsvorgangs, einige nur während des „Linkens“, andere zur Ausführungszeit und wieder andere die ganze Zeit über. Im Komponentendiagramm haben die Darstellungen Typencharakter.

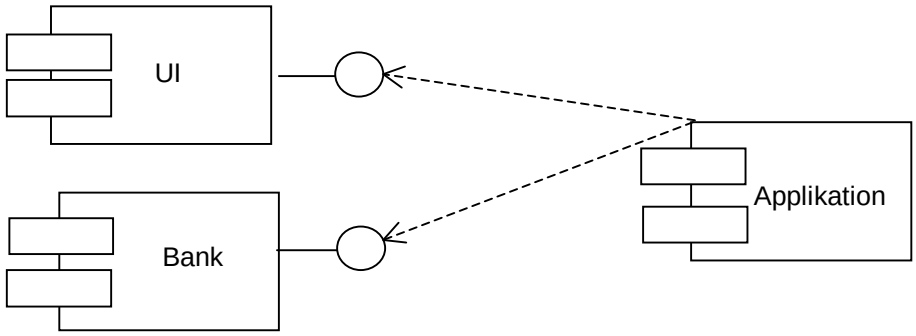


Abb.: Beipiel für ein Komponentendiagramm

Die Komponenten werden als drei ineinander verschachtelte Rechtecke gezeichnet: ihre Schnittstellen sind Kreise am Ende. Das Diagramm enthält ferner Abhängigkeiten in Form von gestrichelten Pfeilen.

2. Einsatzdiagramm

Ein Einsatzdiagramm (deployment diagram) zeigt die Konfiguration der im Prozeß befindlichen Knoten zur Laufzeit sowie der auf ihnen existierenden Komponenten. Knoten (Quader) stellen eine Hardware- oder Verarbeitungseinheit dar. Unter den Konten existieren Verbindungen. Dabei handelt es sich um die physikalischen Kommunikationspfade, die als Linien gezeichnet werden.

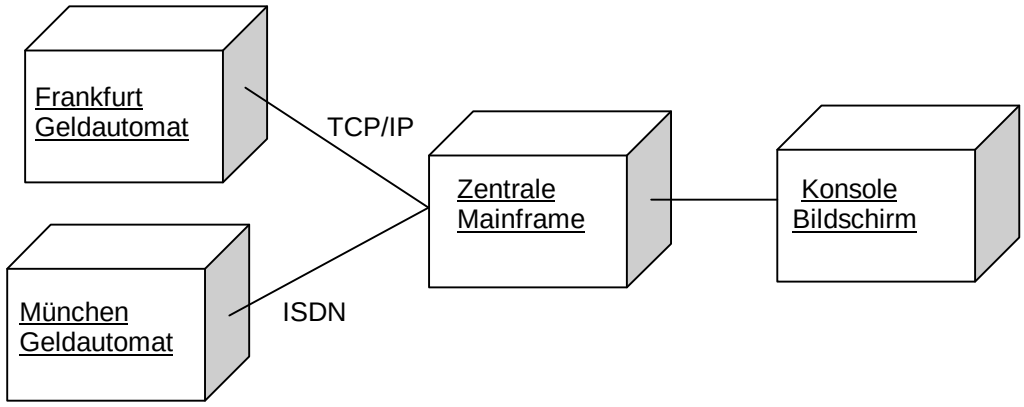


Abb.: Deployment-Diagramm

1.3.8.2 Schema-Modellierung

Die Klassendiagramme der UML bilden eine Obermenge von Entity-Relationship-Diagrammen. Klassendiagramme lassen auch das Modellieren von Verhaltensweisen zu. In einer physischen Datenbank werden diese logische Operationen im allg. in Trigger oder gespeicherte Prozeduren umgewandelt.

1. Modellierung eines logischen Datenbankschemas

Das Modellieren eines logischen Datenbankschemas[42] umfaßt:
- die Identifikation von Klassen, deren Zustand die Lebensdauer ihrer Anwendungen überdauern muß.
- das Erstellen eines Klassendiagramms, das all diese Klassen enthält, und das Markieren dieser Klassen mit den Standardeigenschaftswert `persistent`.
- Expansion der strukturellen Eigenschaften dieser Klassen (Spezifikation der Details zu den Attributen, Konzentration auf Assoziationen und Kardinalitäten).
- Beachtung typischer Muster, z.B. rekursive Assoziationen, Eins-zu-eins-Assoziationen, n-äre Assoziationen.
- Betrachtung der Vehaltensweisen dieser Klassen, z.B. durch Expansion von Operationen, die für Datenzugriff und Integrität der Daten wichtig sind.
- Überführung, falls möglich mit Hilfe von Programmen, des logischen in einen physischen Entwurf

Bsp.: Modellierung eines Datenbankschemas für eine Fachhochschule

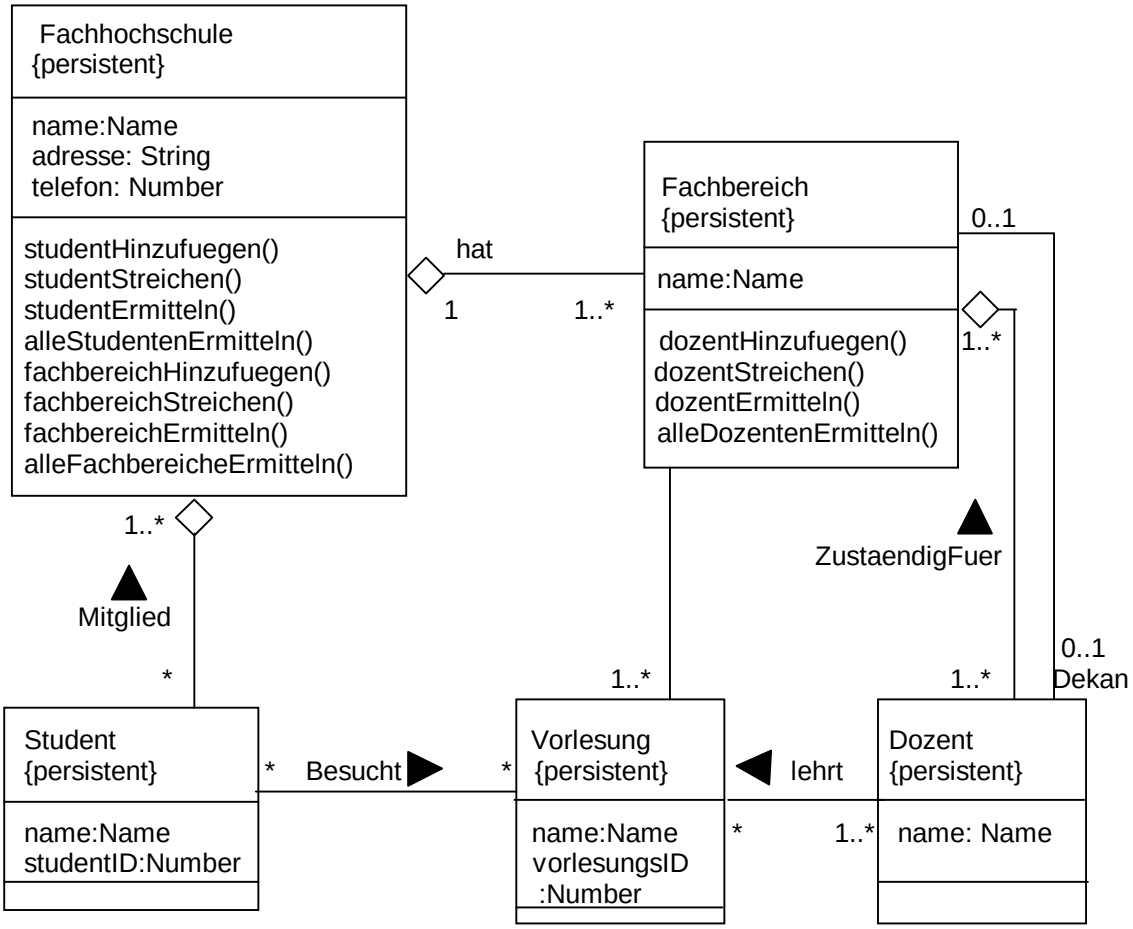


Abb.: Modellieren eines logischen Datenbankschemas

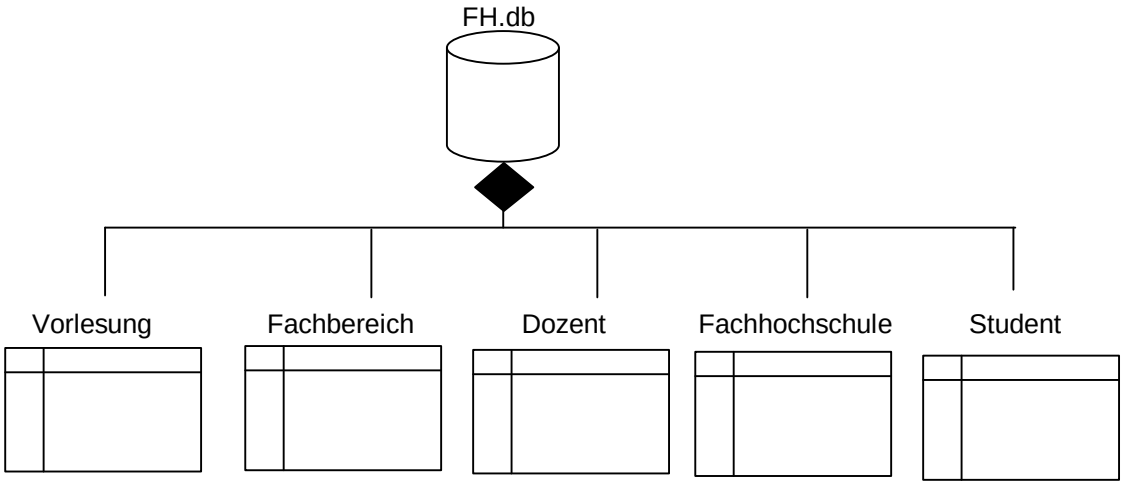
[42] Booch, Grady u. a.: Das UML-Benutzerhandbuch, Addison-Wesley, Bonn 1999, Seite 123

2. Die Modellierung eines physischen Datenbankschemas

Das Modellieren einer physischen Datenbank[43] umfaßt:

- Identifikation der Klassen, die das logische Datenbankschema bilden
- Wahl einer Strategie zur Abbildung dieser Klassen auf Tabbellen.
- Erstellen eines Komponentendiagramms (mit den als Tabellen stereotypisierten Tabellen) zur Visualisierung und Dokumentation

Bsp.: Modellieren der physischen Datenbank FH.db



[43] vgl. Booch, Grady u.a.: Das UML-Benutzerhandbuch, Seite 451.

1.4 Standards

Auf dem DB-Markt existieren Systeme verschiedener Hersteller, die in ihren Strukturen, ihrer Terminologie und vor allem in den Sprachen erhebliche Unterschiede aufweisen. Schrittweise wurden zur Behebung dieses Zustandes von verschiedenen Gremien (DIN, ANSI, CODASYL, ISO) allgemeine DB-Standards empfohlen und Normenvorschläge unterbreitet. Die Empfehlungen weisen teilweise bzgl. Funktionsumfang und Realisierungsaufwand erhebliche Unterschiede auf, sind aber dennoch bedeutende Eckpfeiler der DB-Geschichte.
Von großer Bedeutung sind auch die durch die Position des Marktführeres IBM verbreiteten "defacto"-Standards (z.B. IMS, DL/1 oder das System R).

1.4.1 CODASYL-Konzept

Die "Conference on Data Systems Languages (**CODASYL**)" ist ein Ausschuß von Herstellern und großen amerikanischen Anwendern, der sich um die Normung von Programmiersprachen bemüht. Auch im Bereich der Datenbanken wurde die CODASYL-Gruppe aktiv. Gegen Ende der 60er Jahre wurde die "Data Base Task Group (**DBTG**)" beauftragt, einen Standardisierungsvorschlag für das bereits weit auseinanderlaufende Feld der Datenbankentwicklung auszuarbeiten.

Die wichtigsten Empfehlungen der Ausarbeitung waren:

1. Definition der Syntax einer Datenbank-Beschreibungssprache (**DDL**)
2. Definition der Syntax einer Datenbank-Manipulationssprache (**DML**)

Darüber hinaus wurde in einer "Feature Analysis" ein Modell-DBMS konzeptiert. Dieses Konzept, von führenden DB-Fachleuten erarbeitet, steht allen Herstellern und Anwendern zur Verfügung. Seine externen Schnittstellen sind exakt beschrieben, die interne Realisierung bleibt jedem Hersteller selbst überlassen. CODASYL-DB werden bzw. wurden angeboten von Honeywell (IDS), Univac (DMS/1100), Cullinane Corp. (IDMS) und Siemens (UDS).

1.4.2 IMS

`IMS` (Integrated Mangement System) ist eines der ältesten Systeme und daher eines der am weitesten entwickelten DB/DC-Systeme. `IMS` wurde bereits Mitte der 60er Jahre konzipiert, die 1. Version Ende 1968 freigegeben. Seit 1973 ist IMS in den virtuellen Systemen (OS/VS1 und OS/VS2) verfügbar (IMS/VS). Darüberhinaus wurden am Markt die Varianten DL/1 DOS/VS und DL/1-Entry angeboten, die funktionell eine Untermenge von IMS/VS darstellen.

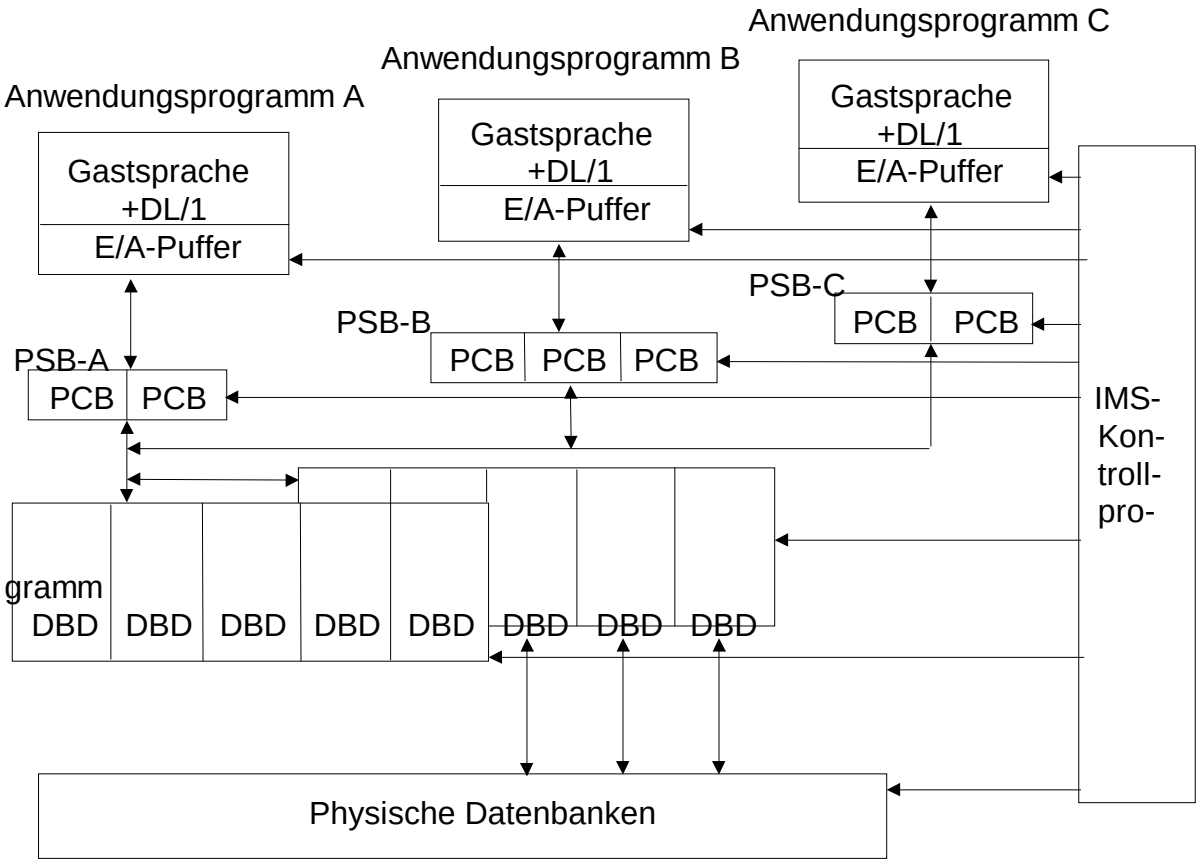


Abb. 1.4-1: Aufbau IMS

Kurzbeschreibung der DB-Struktur von IMS

IMS besteht in der Regel nicht aus einer einzigen, sondern aus mehreren Datenbanken. IMS erlaubt den Aufbau hierarchisch strukturierter Datenbestände, die für das System in 3 verschiedenen Stufen beschrieben werden. Man unterscheidet zwischen der Beschreibung des hierarchischen Datenbankaufbaus, der Be-schreibung des Zugriffs für ein bestimmtes Programm und der Beschreibung der Segmente mit den Datenfeldern. Als Segment bezeichnet man die kleinste Zugriffseinheit, das ist die kleinste Menge an Daten, die durch DL/1-Operationen transportiert werden kann (DL/1 ist die Sprache (DDL, DML)) von IMS. Zur Beschreibung des hierarchischen Aufbaus einer physischen Datenbank dient die „Data Base Description (DBD)“. Eine physische Datenbank ist die tatsächlich gespeicherte Datenbank, im Gegensatz zum Datenbankausschnitt (program communication block, PCB), der dem Benutzer zur Verfügung gestellt wird. Die Beschreibung des speziellen Datenbank-Zugriffs für ein Programm erfolgt im "program specification block (PSP)". Felder und Struktur eines Segments werden im Programm selbst beschrieben, sie entsprechen den Konventionen der jeweiligen Wirtsprache.

1.4.3 System R und SQL

1.4.3.1 System R

Hierbei handelt es sich um den Prototyp[44] relationaler Datenbanken. Die Entwicklung erfolgte in den Forschungslabors der IBM. Aktuell davon ist vor allem der Sprachentwurf SEQUEL (**SQL**) bzw. die "zweidimensionale" Zugriffssprache "Query by Example (**QBE**)"[45]. Die beiden Sprachen sind grundlegend für den Sprachentwurf in relationalen Datenbanken.

1.4.3.2 **Standard-SQL** (Structured Query Language)

Im Februar 1987 wurde SQL zum offiziellen Standard des American National Standard Institute (ANSI). Im März 1987 wurde SQL die Datenbank-Zugriffssprache im Rahmen der System Application Architecture (SAA) von IBM. Inzwischen wurde der "Standard von 1986" bereits zweimal aktualisiert (1989, 1992). Die aktuelle Fassung (SQL-92, aus "taktischen Gründen" SQL2 genannt) umfaßt bereits eine 600 Seiten umfassende Beschreibung.

1. Die **`select`**-Anweisung

- Die zentrale Idee

Die zentrale Idee, die SQL zugrundeliegt und in der **select**-Anweisung verwirklicht wurde, ist eine Abbildung bestimmter Tabellenspalten. Die Tabellen bilden den Definitionsbereich. Über eine Auswahlkriterium wird aus dem Definitionsbereich ein Abbildungsbereich bestimmt:

```
select <spalten>            /* 1. Klausel */
from <tabellen>             /* 2. Klausel */
[where <bedingung> ... ]    /* 3. Klausel */
```

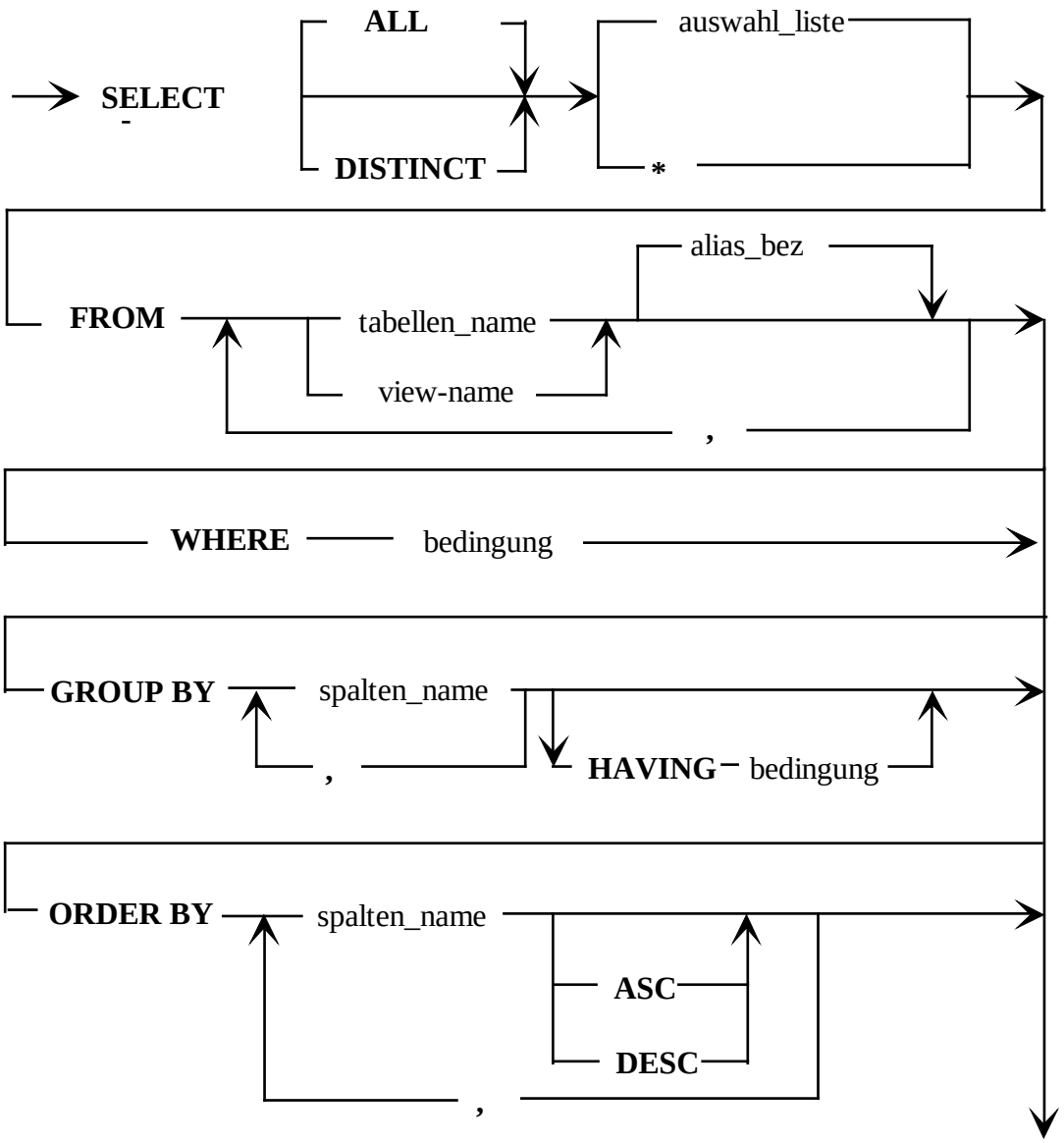
Der 1. Klausel legt fest, welche Spalten (Attribute) im Ergebnis (Abbildungsbereich) berücksichtigt werden sollen. Die 2. Klausel bezeichnet die Tabellen und Views (Sichten), aus denen Zeilen und Spalten abgefragt werden sollen. Die 3. Klausel ist optional und bestimmt, daß nur diejenigen Zeilen im Ergebnis angezeigt werden sollen, die einer bestimmten Bedingung genügen.
Der formale Aufbau des `select`-Befehls ist an bestimmte Regeln gebunden, die die folgenden Syntax-Diagramme[46] für den Standard von 1989 beschreiben:

---

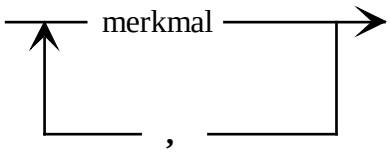
[44] vgl. Sandberg, G.: A primer of relational data base concepts, IBM Systems Journal, Heft 1, S.23 - 61, (1980)

[45] vgl. Zloof, M.M.: Query by Example: a database language, IBM Systems Journal, Heft 4, S. 324 - 342, (1977)

[46] Alle Bezeichnungen in nicht fettgedruckten Kleinbuchstaben werden entweder in anderen Diagrammen erläutert bzw. im Text beschrieben. Großgeschriebene Wörter in Fettdruck bezeichnen Schlüsselwörter bzw. terminale Symbole der select-Grammatik. Sie können unmittelbar genutzt werden. Wörter, die nur aus kleingeschriebenen Buchstaben bestehen, müssen ersetzt werden.

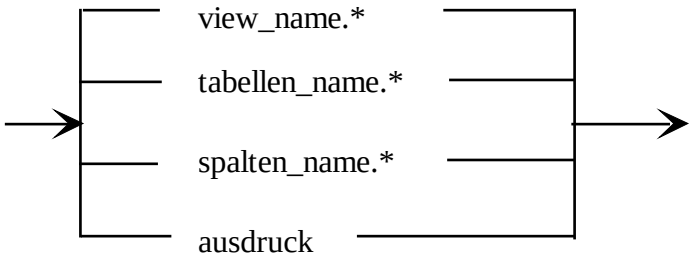


auswahl_liste

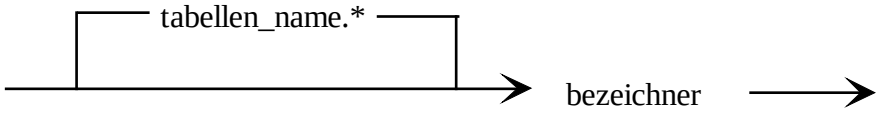


Die hier angegebene Grammatik des select-Befehls bezieht sich auf ENTRY-SQL von SQL-92. Ein nach diesen Vorschriften aufgebauter SQL-Befehl müßte von allen herkömmlichen, auf dem Markt befindlichen relationalen Datenbanksystemen verstanden werden.

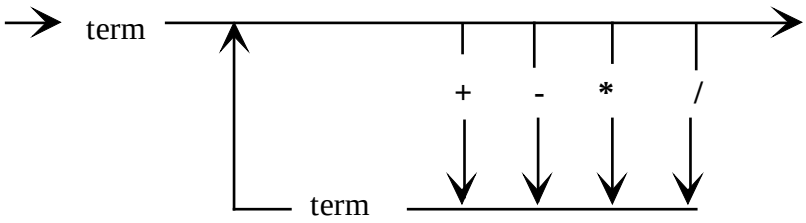
merkmal



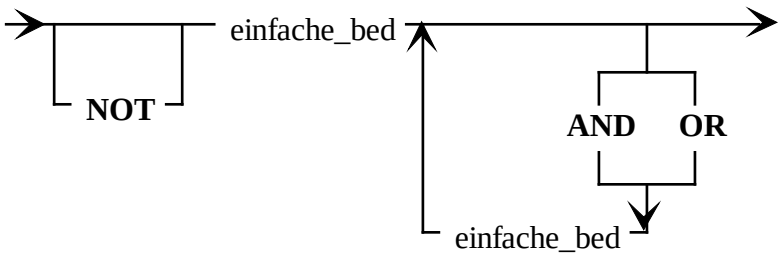
spalten_name



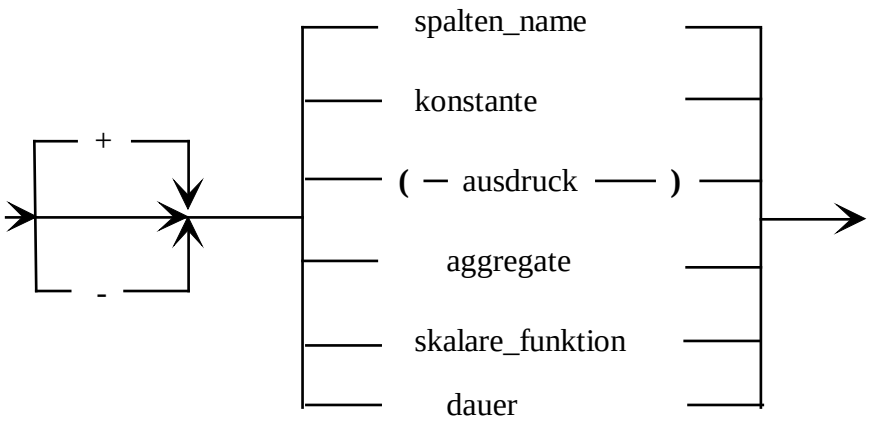
ausdruck



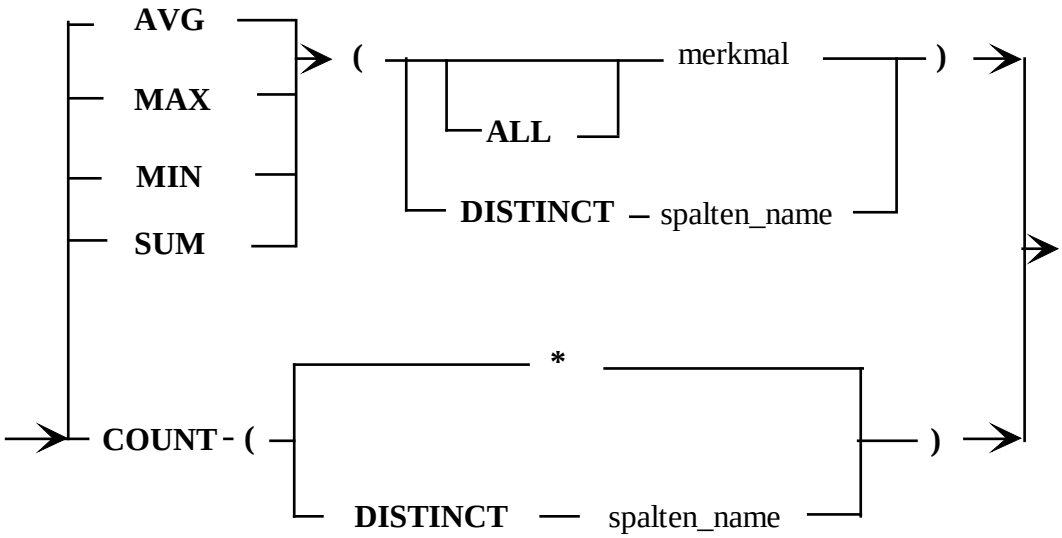
bedingung



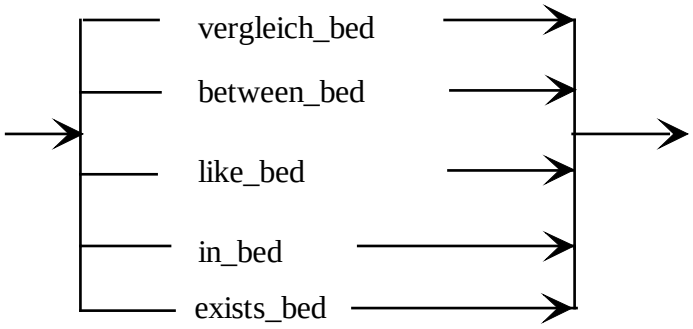
term



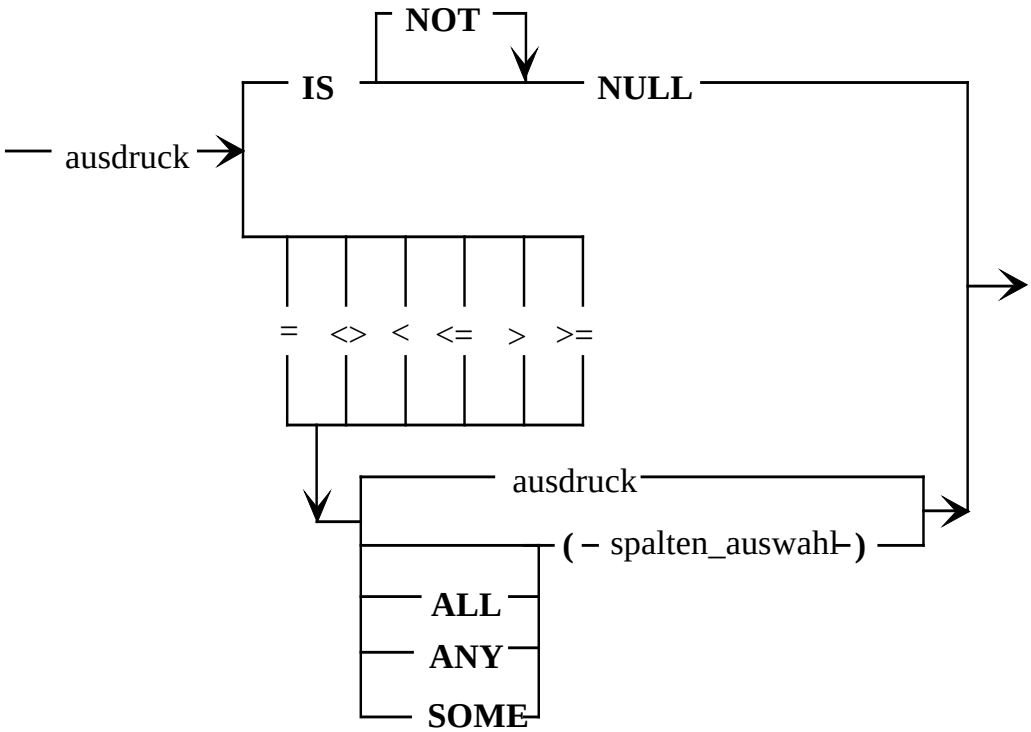
aggregate



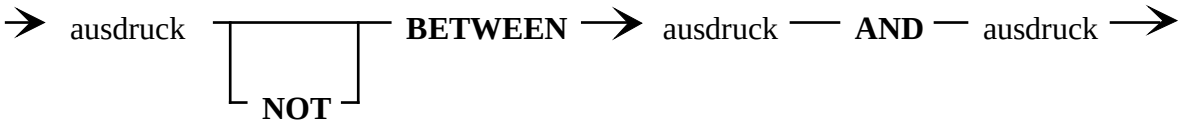
einfache_bed



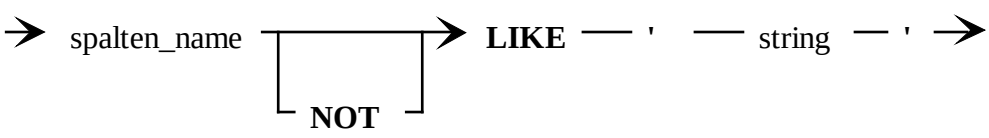
vergleich_bed



between_bed



like_bed

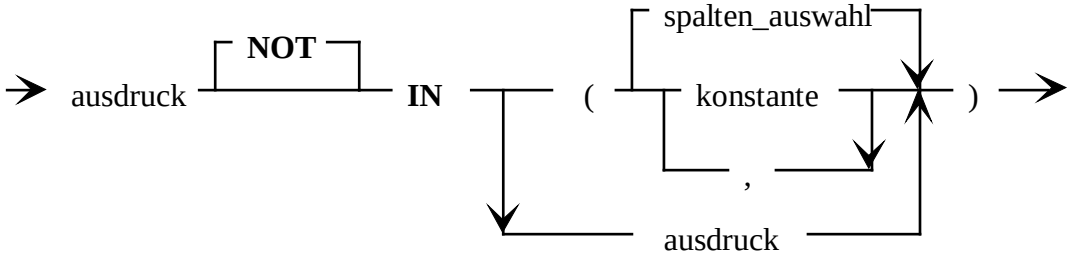


Bei "string" in like_bed" haben "_" und "%" eine spezielle Bedeutung:

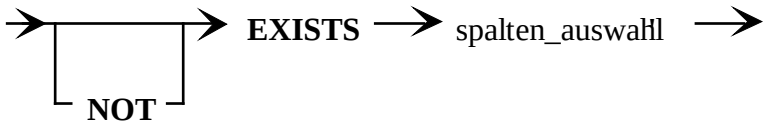
- "_" steht für irgendein Symbol

- "%" steht für irgendeine Folge von 0 oder mehr Zeichen

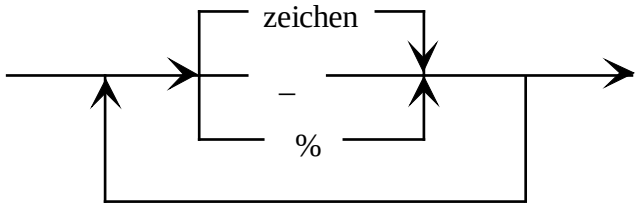
is_bed



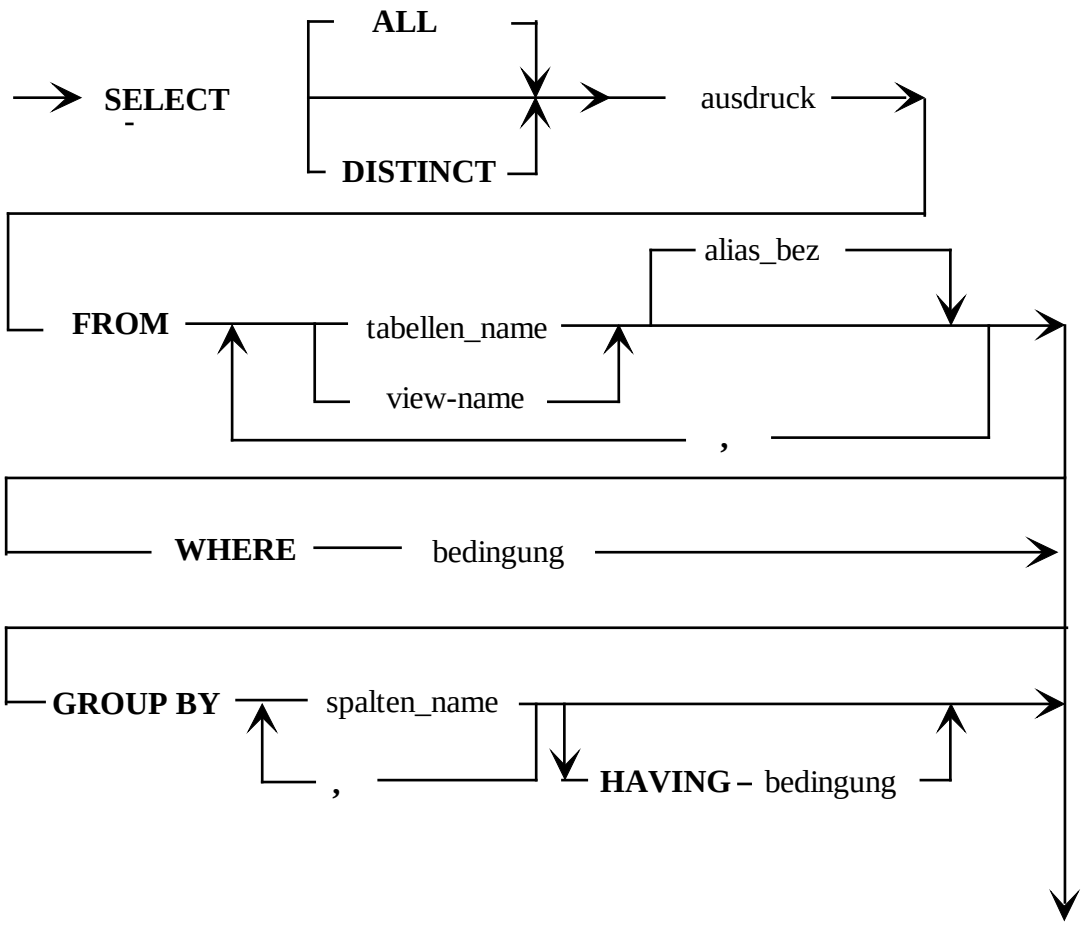
exists_bed



string



spalten_auswahl



union_query

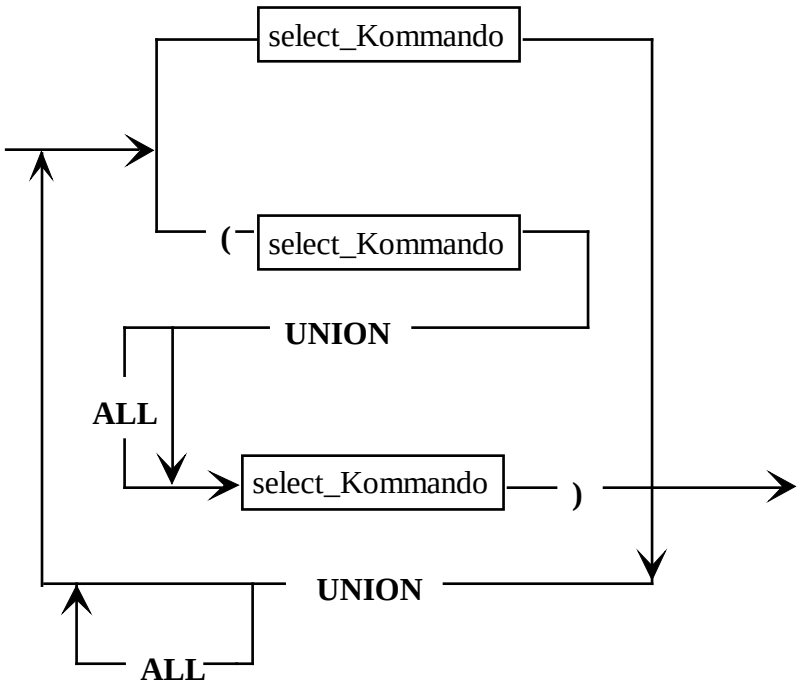


Abb. 1.4-2: Syntaxdiagramme zum SQL-Kommando

- Die Basis der „select“-Anweisung

Jeder SQL-Befehl bzw. jede SQL-Anweisung beginnt mit einem Befehlswort, das die auszuführende Operation bezeichnet, z.B. **select**. Zu vielen SQL-Befehlen gehören eine oder mehrere Klauseln, mit denen der Befehl für eine spezielle Anforderung angepaßt wird. Jeder SQL-Befehl muß zwei Bedingungen erfüllen:

1) Bezeichnen der gewünschten Daten
   (eine Menge von Zeilen in einer oder mehreren Tabellen)
2) Bestimmen, was mit den Daten geschehen soll.

Die Anweisung **select-from-where** gibt dem Benutzer immer eine Resultatstabelle zurück.
Tabellen werden über die sogenannte „leere Bedingung“ aufgelistet, z.B. die Tabelle „`angestellte`“[47]:

```
SELECT *
FROM ANGESTELLTE;[48]
```

Man kann sich auch bei der Auswahl der Ausgabespalten auf bestimmte Felder beschränken:

```
SELECT ANG_ID, NAME, GEBDAT
FROM ANGESTELLTE;
```

Die ausgewählten Datensätze können über die zusätzliche (optionale) `ORDER BY` - Klausel in eine sortierte Reihenfolge gebracht werden:

```
SELECT *
FROM ANGESTELLTE
ORDER BY NAME;
```

Die Sortierreihenfolge kann auf- (`ASC`) oder absteigend (`DESC`) sein. Auch sekundäre bzw. tertiäre Sortierkriterien können gebildet werden:

```
SELECT *
FROM ANGESTELLTE
ORDER BY ANG_ID ASC, ABT_ID DESC;
```

Wiederholungen von Tabellenspalten-Werten in der Ausgabe werden durch Angabe von `DISTINCT` vor dem Feldnamen ausgeschlossen.
Eine andere konkrete Anfrage an die vorliegende Tabelle könnte sein:

```
SELECT * FROM ANGESTELLTE
WHERE NAME = 'Fritz';
```

SQL kennt 6 relationale Operatoren zur Angabe von Bedingungen: „ `=, <>` `oder !=, <, >, <=, >=`“

Bsp.: Welcher Angestellte ist vor dem 2. Dezember 1957 geboren?

---

[47] Vgl. 1.3.3
[48] SQL ist nicht `case sensitive`

```
SELECT * FROM ANGESTELLTE
WHERE GEBDATUM < '02-DEC-57'
ORDER BY NAME;
```

- Komplexe Bedingungen

Mehrere Suchbedingungen können in einer WHERE-Klausel kombiniert werden, z.B.

```
SELECT * FROM ANGESTELLTE
WHERE NAME = ‘Fritz‘ AND JOB_ID = ‘SY‘;
```

Drei logische Operatoren **NOT**, **AND**, **OR** ermöglichen die logische Kombination von Suchbedingungen. Die angegebene Reihenfolge entspricht der Priorität ihrer Verarbeitung. Mit Klammern kann die Abarbeitungsfolge beeinflußt werden.

- IN, BETWEEN und LIKE

SELECT kann auch in Zusammenhang mit den Operatoren **BETWEEN**, **LIKE** und **IN** Suchbedingungen über die WHERE-Klausel festlegen. Die 2 Operatoren BETWEEN, LIKE ersetzen relationale Operatoren und legen den Geltungsbereich der WHERE-Klausel fest
Der mit **BETWEEN** festgelegte Bereich ist inklusiv: Auch Zeilen, deren Feldwert einem der beiden Grenzwerte entspricht, werden in der Ergebnistabelle aufgenom-men, genauso wie alle dazwischenliegenden Werte.

```
SELECT * FROM ANGESTELLTE
WHERE GEBDATUM BETWEEN '01-JAN-55' AND '31-DEC-55';
```

Der Operator **IN** prüft, ob ein Feldwert mit einem Wert in der angegebenen Werteliste übereinstimmt:

```
SELECT ABT_ID, BEZEICHNUNG
FROM ABTEILUNG
WHERE ABT_ID IN ('KO', 'OD')
ORDER BY ABT_ID;
```

LIKE ermöglicht die Suche nach Zeichenfolgen. Dabei ersetzt

- der Unterstrich (_) ein beliebiges einzelnes Zeichen
- das Prozentzeichen (%) eine beliebige Anzahl von Zeichen

Bsp.: Welche Angestellte haben einen Namen der mit 'F' beginnt?

```
SELECT * FROM ANGESTELLTE
WHERE NAME LIKE 'F%';
```

- Die Bedingung „IS NULL“

Zur Ermittlung von Tupeln, die NULL-Werte in einem Attribut besitzen, gibt es den speziellen Operator **IS NULL** bzw. IS NOT NULL. NULL-Werte sind nicht durch

Zeichen repräsentiert, daher ist der Operator auch nicht gleichwertig mit einem "=". Bei einer Selektion durch eine ORDER BY-Klausel erscheinen NULL-Werte stets am Anfang (unabhängig von der gewählten Sortierfolge).

- Verbund

Es besteht grundsätzlich die Möglichkeit mehrere Tabellen im Rahmen von Verbundoperationen zu verknüpfen.

Bsp.: Welche Angestellten üben den Beruf 'Systemplaner' aus?

```
SELECT ANGESTELLTE.ANG_ID, ANGESTELLTE.NAME, JOB.TITEL
FROM ANGESTELLTE, JOB
WHERE ANGESTELLTE.JOB_ID = JOB.JOB_ID AND JOB.TITEL = 'Systemplaner';
```

Bsp.: Welchen Beruf übt welcher Angestellte aus?

```
SELECT ANGESTELLTE.ANG_ID, ANGESTELLTE.NAME, JOB.TITEL
FROM ANGESTELLTE, JOB
WHERE ANGESTELLTE.JOB_ID = JOB.JOB_ID;
```

Bsp.: Bestimme alle Angestellten aufsteigend geordnet nach Berufen (job.titel) und innerhalb des Berufs absteigend nach Gehalt (job.gehalt). Die Ausgabe soll in eine Tabelle mit folgenden Spaltenüberschriften erfolgen:

```
job.titel   job.gehalt    angestellte.name
```

```
SELECT JOB.TITEL, JOB.GEHALT, ANGESTELLTE.NAME
FROM JOB, ANGESTELLTE
WHERE JOB.JOB_ID = ANGESTELLTE.JOB_ID
ORDER BY JOB.TITEL, JOB.GEHALT DESC;
```

Über die SELECT-Anweisung kann eine Tabelle mit sich selbst verknüpft werden (Selbstverknüpfung, SELF JOIN). Mit Selbstverknüpfungen können Daten aus verschiedenen Zeilen einer einzigen Tabelle zueinander in Beziehung gesetzt werden, und diese Daten in einer Ergebnistabelle zu Zeilen zusammengefaßt werden. Der SELF JOIN kann über die Definition unterschiedlicher Namen für eine Tabelle realisiert werden. SQL behandelt die unter verschiedenen Namen geführten Tabelle wie unterschiedliche, eigenständige Tabellen.

Bsp.: Bestimme aus der Tabelle Angestellte, jeweils für einen Mitarbeiter die Angestellten, mit denen er in der gleichen Abteilung arbeitet. Die Ausgabe soll in eine Tabelle mit folgenden Spaltenüberschriften erfolgen:

```
A.ANG_ID  A.NAME  B.ANG_ID  B.NAME A.ABT_ID
```

```
SELECT A.ANG_ID, A.NAME, B.ANG_ID, B.NAME, A.ABT_ID
FROM ANGESTELLTE A, ANGESTELLTE B
WHERE A.ABT_ID = B.ABT_ID AND
A.NAME <> B.NAME
ORDER BY A.NAME;
```

Für den Verbund einer Tabelle mit sich selbst wird genutzt, daß jeder Name einer Tabelle eine (gewissermaßen selektionsinterne) Abkürzung (alias) erhalten

kann (durch Leerzeichen getrennt, dem Tabellennamen in der `from`-Klausel nachgestellt).

- Unterabfragen

Bedingungen in `SELECT`- (`DELETE`- oder `UPDATE`-) Kommandos sowie `HAVING`-Klauseln dürfen sogenannte "**Subqueries**" (**Unterabfragen**) enthalten. In diesem Fall wird ein Wert bzw. eine Menge von Werten in einer Bedingung errechnet. Das Ergebnis dieser Berechnung wird direkt in die befreffende Anfrage eingesetzt, jedoch nicht gespeichert.
Der Aufbau der übergeordneten "`select`"-Anweisung hängt ab von der Anzahl der Werte, die als Ergebnis der untergeordneten `SELECT`-Abfrage ausgegeben werden. Folgende Fälle sind möglich:

- Unterabfrage mit einem Ergebnis
- Unterabfrage mit mehreren Ergebnissen

Eine Unterabfrage mit einem Ergebnis ist:

```
SELECT JOB.TITEL, JOB.GEHALT
FROM JOB
WHERE JOB.JOB_ID =
 (SELECT ANGESTELLTE.JOB_ID FROM ANGESTELLTE
  WHERE ANGESTELLTE.NAME = 'Fritz');
```

Bsp.: Bestimme alle Angestellten, die diesselbe Funktion (Titel) wie ‘Fritz‘ haben oder in der gleichen Abteilung beschäftigt sind

```
SELECT NAME FROM ANGESTELLTE
WHERE ANGESTELLTE.JOB_ID =
 (SELECT ANGESTELLTE.JOB_ID FROM ANGESTELLTE, JOB
  WHERE  NAME = 'Fritz' AND ANGESTELLTE.JOB_ID = JOB.JOB_ID)
OR ABT_ID =
  (SELECT ABT_ID FROM ANGESTELLTE WHERE NAME = 'Fritz')
ORDER BY NAME;
```

In Unterabfragen mit mehr als einem Ergebnis muß eine "`where`"-Klausel verwendet werden, die mehr als einen Wert akzeptiert. Eine solche Abfrage kann die Operatoren **IN**, **ANY**, **ALL** verwenden, z.B.:

1) Welche Angestellten üben die Tätigkeiten eines Systemplaners bzw. eines Ingenieurs aus?

```
SELECT ANGESTELLTE.NAME
FROM ANGESTELLTE
WHERE ANGESTELLTE.JOB_ID IN
 (SELECT JOB.JOB_ID
  FROM JOB
  WHERE JOB.TITEL = 'Systemplaner' OR
        JOB.TITEL = 'Ingenieur');
```

bzw.

```
SELECT ANGESTELLTE.NAME
FROM ANGESTELLTE
```

```
WHERE ANGESTELLTE.JOB_ID = ANY
 (SELECT JOB.JOB_ID
  FROM JOB
  WHERE JOB.TITEL = 'Systemplaner' OR
        JOB.TITEL = 'Ingenieur');
```

2) Finde alle Mitarbeiter (Angestellte) in der Abteilung `'Konstruktion'`, die einen Beruf (`job.titel`) haben, der zu einem Mitarbeiter der Abteilung `'Organisation und Datenverarbeitung'` gleich ist.

```
SELECT ANGESTELLTE.NAME, JOB.TITEL
FROM ANGESTELLTE, JOB, ABTEILUNG
WHERE ABTEILUNG.BEZEICHNUNG = 'KONSTRUKTION' AND
      ABTEILUNG.ABT_ID = ANGESTELLTE.ABT_ID  AND
      ANGESTELLTE.JOB_ID = JOB.JOB_ID        AND
      JOB.TITEL IN
      (SELECT JOB.TITEL FROM JOB, ANGESTELLTE, ABTEILUNG
       WHERE ABTEILUNG.BEZEICHNUNG = 'Organisation und Datenverarbeitung'
             AND ANGESTELLTE.JOB_ID = JOB.JOB_ID);
```

3) Finde alle Mitarbeiter (Angestellte) in der Abteilung `'Konstruktion'`, die einen Beruf (`job.titel`) haben, der zu einem Mitarbeiter der Abteilung `'Organisation und Datenverarbeitung'` nicht gleich ist.

```
SELECT ANGESTELLTE.NAME, JOB.TITEL
FROM ANGESTELLTE, JOB, ABTEILUNG
WHERE ABTEILUNG.BEZEICHNUNG = 'Konstruktion' AND
      ABTEILUNG.ABT_ID = ANGESTELLTE.ABT_ID  AND
      ANGESTELLTE.JOB_ID = JOB.JOB_ID        AND
      JOB.TITEL NOT IN
      (SELECT JOB.TITEL FROM JOB, ANGESTELLTE, ABTEILUNG
       WHERE ABTEILUNG.BEZEICHNUNG = 'Organisation und Datenverarbeitung'
             AND ANGESTELLTE.JOB_ID = JOB.JOB_ID);
```

4) Finde alle Angestellten, die mehr verdienen als irgendein Mitarbeiter in der Personalabteilung. Die Ausgabe soll in eine Tabelle mit folgenden Spaltenüberschriften erfolgen:

```
 JOB.GEHALT  JOB.TITEL   ANGESTELLTE.NAME   ANGESTELLTE.ABT_ID
```

```
SELECT JOB.GEHALT, JOB.TITEL, ANGESTELLTE.NAME, ANGESTELLTE.ABT_ID
FROM ANGESTELLTE, JOB
WHERE JOB.GEHALT > ANY
(SELECT GEHALT FROM JOB, ANGESTELLTE, ABTEILUNG
 WHERE ABTEILUNG.BEZEICHNUNG = 'PERSONALABTEILUNG'
 AND ABTEILUNG.ABT_ID = ANGESTELLTE.ABT_ID
 AND ANGESTELLTE.JOB_ID = JOB.JOB_ID)
AND JOB.JOB_ID = ANGESTELLTE.JOB_ID
ORDER BY GEHALT DESC;
```

Die Vergleichs-Operatoren einer "vergleich_bed"[49] dürfen in Unterabfragen (Subquery) durch ALL, ANY oder SOME (Synonym für ANY) modifiziert werden. Eine Anfrage darf mehrere "Subqueries" enthalten, Unterabfragen dürfen geschachtelt sein. Es ist möglich, "IN" für “= ANY" und "NOT IN" für "!= ALL" zu setzen. Die "ORDER-BY"-Klausel ist in Unterabfragen nicht zugelassen.

[49] Vgl. Syntaxdiagramme Abb. 1.4-2

- EXISTS

Unterabfragen können auch mit dem Operator **EXISTS** verknüpft werden. Der Ausdruck "`exists (select ..... )`" führt zum logischen Wert "wahr", falls das Ergebnis des in Klammern eingeschlossenen "`select`-"Ausdrucks nicht leer ist.

```
SELECT JOB.TITEL FROM JOB
WHERE EXISTS
 (SELECT * FROM ANGESTELLTE
 WHERE ANGESTELLTE.JOB_ID = JOB.JOB_ID)
ORDER BY JOB.TITEL;
```

Über **EXISTS** kann der Existenzquantor in SQL interpretiert werden. SQL hat den Allquantor nicht implementiert. Allerdings können Abfragen mit dem Allquantor immer mit Hilfe des negierten Existenzquantors formuliert werden. Es gibt komplexe Unterabfragen, die die Verwendung von EXISTS in negierter Form erforderlich machen
Bsp.: Wähle die Angestellten aus, die die Qualifikation 'Systemplaner' haben

```
SELECT * FROM ANGESTELLTE
WHERE NOT EXISTS
(SELECT * FROM JOB
 WHERE JOB.TITEL <> 'Systemplaner' AND
       JOB.JOB_ID = ANGESTELLTE.JOB_ID);
```

Die vorliegende Abfrage zeigt eine Anwendung mit dem Allquantor der Prädikatenlogik. Standard-SQL hat den Allquantor nicht implementiert. Abfragen mit dem Allquantor können mit Hilfe des negierten Existenzquantors formuliert werden, z.B.: „Wähle die Qualifikation so aus, daß keiner existiert, der nicht die Qualifikation ‘Systemplaner‘ (SY) besitzt“.

- Aggregatfunktionen

Aggregat-Funktionen berechnen tupelübergreifend Summe, Durchschnitt, Aufzählungen. So zählt die Funktion `COUNT(*)` alle Tupel einer Tabelle (einschl. Duplikate und einschl. aller Tupel, die Nullwerte enthalten).

```
SELECT COUNT(*)
FROM ANGESTELLTE;
```

Das Symbol **`(*)`** bestimmt, daß sich die Funktion `COUNT()` auf die gesamte Zeile bezieht, nicht auf ein einziges Feld. SQL-Aggregat-Funktionen sind:

COUNT()
liefert die Anzahl der ausgewählten Zeilen
SUM()
liefert die Summe der Werte in einem numerischen Feld
MIN()
liefert den kleinsten Wert einer Zeichen- ,Datums- oder einer numerischen Spalte
MAX()
liefert den größten Wert einer Zeichen-, Datums- oder einer numerischen Spalte

AVG()
liefert den Mittelwert (Durchschnitt) der Werte eines numerischen Feldes (Spalte).

Aggregat-Funktionen können auch mit der WHERE-Klausel kombiniert werden, z.B.:

1) Stelle eine Liste von Angestellten zusammen, die am meisten verdienen

```
SELECT ANGESTELLTE.ANG_ID, ANGESTELLTE.NAME
FROM ANGESTELLTE, JOB
WHERE ANGESTELLTE.JOB_ID = JOB.JOB_ID AND
 JOB.GEHALT = (SELECT MAX(JOB.GEHALT) FROM JOB);
```

2) Welcher Angestellte hat ein monatliches Einkommen, das über dem Durchschnittseinkommen liegt?

```
SELECT A.ANG_ID, A.NAME
FROM ANGESTELLTE A, JOB J
WHERE A.JOB_ID = J.JOB_ID AND
J.GEHALT > (SELECT AVG(JOB.GEHALT) FROM JOB);
```

Aggregat-Funktionen können mit der GROUP-BY-Klausel auf disjunkte Teilmengen einer Tupelmenge bezogen werden, z.B.:

1) Wieviel Angestellte sind in der Abteilung OD beschaeftigt?

```
SELECT ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG, COUNT(ANGESTELLTE.ANG_ID)
FROM ABTEILUNG, ANGESTELLTE
WHERE ABTEILUNG.ABT_ID = 'OD' AND ANGESTELLTE.ABT_ID = 'OD'
GROUP BY ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG;
```

2) Bestimme eine Liste, die die Identifikation, Bezeichnung einer Abteilung und die Zahl der Beschaeftigten in dieser Abteilung zeigt.

```
SELECT ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG, COUNT(ANGESTELLTE.ANG_ID)
FROM ABTEILUNG, ANGESTELLTE
WHERE ANGESTELLTE.ABT_ID = ABTEILUNG.ABT_ID
GROUP BY ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG;
```

- GROUP BY und HAVING

Die Anweisung **GROUP BY** ermöglicht es, Datensätze (Tupel) gruppenweise zu ordnen. Auf diese Gruppen lassen sich verschiedene Operationen (in der Regel mit Aggregatsfunktionen) ausführen. Die Klauseln GROUP BY und HAVING bewirken, daß sich diese Funktionen nicht mehr auf alle Zeilen einer Tabelle, sondern auf alle Zeilen innerhalb einer Gruppe auswirken.
Die **GROUP BY**-Klausel führt Zeilen aus der Ergebnistabelle einer SELECT-Anweisung zu Gruppen zusammen, in denen bestimmte Spalten den gleichen Wert haben. Jede Gruppe wird in der Ergebnistabelle zu einer einzigen Zeile reduziert.
Somit kann über **GROUP BY** projiziert werden auf

- Attribute, über die "gruppiert" wird
- Gruppen-Funktionen (, angewendet auf Attribute der Haupttabelle)

- Konstante

Mit der **HAVING**-Klausel kann der Anwender den einzelnen Gruppen bestimmte Bedingungen auferlegen. Sie ist eine Selektion von Gruppen. Da im Ausgangspunkt der HAVING-Klausel bereits Gruppen gebildet sind, kann nur nach Attributen dieser Zwischen-Ergebnistabelle selektiert werden. Mit Hilfe der HAVING-Klausel können Gruppen ausgewählt werden, die in die Ergebnistabelle aufgenommen werden sollen

```
SELECT ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG, COUNT(*)
FROM ABTEILUNG, ANGESTELLTE
WHERE ANGESTELLTE.ABT_ID = ABTEILUNG.ABT_ID
GROUP BY ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG
HAVING COUNT(*) > 1;
```

Die HAVING-Klausel wird erst nach der Gruppierung ausgewertet, denn sie schränkt nur die durch die Gruppierung erzeugten Datenmangen ein. Sie ist aber hier, obwohl eine WHERE-Klausel vor allen anderen Klauseln ausgeführt wird, unbedingt erforderlich, da

```
SELECT ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG, COUNT(*)
FROM ABTEILUNG, ANGESTELLTE
WHERE ANGESTELLTE.ABT_ID = ABTEILUNG.ABT_ID
 AND COUNT(*) > 1
GROUP BY ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG;
```

zu folgender Meldung führt: `GROUP FUNCTION NOT ALLOWED HERE`
Gruppen-Funktionen gehören niemals in ein Prädikat der `WHERE`-Klausel.

- UNION

Häufig möchte man Ergebnisse mehrerer Abfragen verknüpfen. Das kann mit Hilfe des Operators `UNION` geschehen.
Bsp.: Welche Job-Identifikationen gibt es in Angestellte oder Job?

```
SELECT JOB_ID
FROM ANGESTELLTE
UNION
SELECT JOB_ID
FROM JOB;
```

2. Erzeugen bzw. Löschen von Tabellen, Views (Sichten) und der Indexe

Tabellen werden über die Anweisung **CREATE TABLE** ... erzeugt, z.B.:

```
CREATE TABLE ABTEILUNG
 (ABT_ID CHAR(2) NOT NULL,
 BEZEICHNUNG CHAR(40));

CREATE TABLE JOB
 (JOB_ID CHAR(2) NOT NULL,
 TITEL CHAR(30),
 GEHALT DECIMAL(8,2));

CREATE TABLE ANGESTELLTE
```

```
 (ANG_ID CHAR(3) NOT NULL,
 NAME CHAR(10),
 GEBDATUM DATE,
 ABT_ID CHAR(2),
 JOB_ID CHAR(2));

CREATE TABLE QUALIFIKATION
(ANG_ID CHAR(3),
 JOB_ID CHAR(2));
```

Jede Tabellenspalte umfaßt Werte einer bestimmten Domäne. Domänen lassen sich über Angabe des Datentyps der Spaten und der Anzahl Zeichen definieren. Generell unterscheidet SQL vier Basistypen

Datentyp	Erläuterung
char(<Laenge X>)	Zur Speicherung von Zeichenketten der Länge X
Integer	Zur Speicherung von ganzen Zahlen
decimal(<Laenge X, Dezimalstellen Y>)	Zur Speicherung von Zahlen im Festkommaformat. Das Festkommaformat kann insgesamt aus X Stellen und Y Stellen nach dem Dezimalpunkt bestehen
date	Die Datumsangaben sind in den Datenbanksystemen spezifisch formatiert

In `CREATE TABLE ..` ist nach dem Datentyp die Angabe `NULL` (Standardmäßiger Default-Wert) bzw. `NOT NULL` möglich. Damit wird festgelegt, ob eine Spalte `NULL`-Werte (d.h. keine Werte) enthalten darf oder nicht. Primärschlüssel sollten grundsätzlich mit der Option `NOT NULL` ausgestattet sein. `NULL`-Werte werden in allen alphanumerischen Datentypen durch Leer-Strings der Länge 0 repräsentiert. Indexe können mit `CREATE INDEX ...` erstellt werden, z.B.:

```
CREATE UNIQUE INDEX ABT_IDX
ON ABTEILUNG (ABT_ID);
CREATE UNIQUE INDEX JOB_IDX
ON JOB (JOB_ID);
CREATE UNIQUE INDEX ANGEST_IDX
ON ANGESTELLTE (ANG_ID);
CREATE UNIQUE INDEX QUAL_IDX
ON QUALIFIKATION (ANG_ID,JOB_ID);
```

SQL verfügt über zwei Zugriffsmethoden: den sequentiellen und indexorientierten Zugriff. Beim sequentiellen Zugriff beginnt das System am Anfang der Tabellen zu suchen und arbeitet Satz für Satz durch die Tabelle, bis der gewünschte Datensatz gefunden ist. Bei umfangreichen Datenbeständen sollte man für jeden Satz einen Suchbegriff (Index) vereinbaren, der in einer Indextabelle abgespeichert wird. Der Zugriff auf die tatsächlich vorliegende Tabelle kann über die Indextabelle erfolgen. jedem Suchbegriff ist eine eindeutige Positionsangabe (= Satzzeiger) des Datensatzes auf dem externen Speicher zugeordnet.
Für Indexe gelten die folgenden Bearbeitungsregeln:

- Eine „`order by`“-Klausel in einer „`select`“-Anweisung kann jede Spalte einer Tabelle referenzieren – unabhängig davon, ob die Tabelle über einen auf dieser Spalte basierenden Index verfügt oder nicht.
- Ein Index kann sich nicht aus mehr als 16 Spalten zusammensetzen
- Ein Index speichert keine `NULL`-Werte
- SQL verfügt über keine Anweisung mit der der Inhalt eines Indexes überprüft werden kann.
- Ein Index kann nur für eine Tabelle, jedoch nicht für eine Datensicht (view) erstellt werden.

Tabellen bzw. Sichten (views) können auch durch Auswahl aus eine anderen Tabelle erstellt werden (`create table` ... bzw. `create view` ... mit einer Unterabfrage).

Bsp.: Erzeugen einer Tabelle „ABTKOSTEN“

```
CREATE TABLE ABTKOSTEN
(ABT_ID NOT NULL, BEZEICHNUNG NOT NULL,GEHAELTER)
AS SELECT DISTINCT ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG,
          SUM(JOB.GEHALT)
   FROM ABTEILUNG, ANGESTELLTE, JOB
   WHERE ABTEILUNG.ABT_ID = ANGESTELLTE.ABT_ID AND
         JOB.JOB_ID = ANGESTELLTE.JOB_ID
   GROUP BY ABTEILUNG.ABT_ID, ABTEILUNG.BEZEICHNUNG;
```

Eine Datensicht (**view**) ist eine (gespeicherte) Abfrage, deren Ergebnis auf einer Abfrage einer oder mehrerer Tabellen beruht und gespeichert ist.

Bsp.: Erstelle eine Sicht Einkommen, die Identifikation, Name, Abteilungsbezeichnung, Gehalt des Angestellten enthält

```
CREATE VIEW EINKOMMEN(ANG_ID, NAME, BEZEICHNUNG, GEHALT)
AS SELECT ANGESTELLTE.ANG_ID, ANGESTELLTE.NAME,
     ABTEILUNG.BEZEICHNUNG, JOB.GEHALT
FROM ANGESTELLTE, ABTEILUNG, JOB
WHERE ANGESTELLTE.ABT_ID = ABTEILUNG.ABT_ID AND
      ANGESTELLTE.JOB_ID = JOB.JOB_ID;
```

```
SELECT * FROM EINKOMMEN;

ANG NAME       BEZEICHNUNG                                GEHALT
--- ---------- ---------------------------------------- ---------
A1  Fritz      Organisation und Datenverarbeitung            6000
A2  Tom        Konstruktion                                  6000
A3  Werner     Organisation und Datenverarbeitung            3000
A4  Gerd       Vertrieb                                      3000
A5  Emil       Personalabteilung                             3000
A7  Erna       Konstruktion                                  3000
A8  Rita       Konstruktion                                  3000
A9  Ute        Organisation und Datenverarbeitung            6000
A10 Willi      Konstruktion                                  6000
A12 Anton      Organisation und Datenverarbeitung            6000
A13 Josef      Konstruktion                                  6000
A14 Maria      Personalabteilung                             3000

12 rows selected.
```

Bsp.: Bestimme aus der vorliegenden Sicht "einkommen" mit Hilfe einer SQL-Anweisung eine Liste mit folgender Ausgabe:

- Identifikation, Name, Gehalt des Angestellten
- das durchschnittliche Gehalt, das in der Abteilung verdient wird, der der Angestellte angehört

```
SELECT A.ANG_ID, A.NAME, A.GEHALT, AVG(ALL B.GEHALT)
FROM EINKOMMEN A, EINKOMMEN B
WHERE A.BEZEICHNUNG = B.BEZEICHNUNG
GROUP BY A.ANG_ID, A.NAME, A.GEHALT;
```

Allgemein können über „views“ nicht nur Selektionen ausgeführt werden, sondern auch Einfügungen, Änderungen, Löschungen unter folgenden Bedingungen:

- Bie Löschungen über einen „view“ darf der „view“ keine GROUP BY- oder DISTINCT-Klausel enthalten, und das Pseudo-Attribut ROWNUM nicht verwenden. Anderenfalls könnten zu einem über den „view“ sichtbaren Datensatz in der Tabelle mehrere Datensätze existieren, die wegen der „DISTINCT“-Klausel nicht unterscheidbar wären
- Bei Änderungen über einen „view“ darf der Änderungswert nicht über einen Ausdruck bestimmt werden. Es gelten diesselben Bedingungen wie beim Löschen.
- Beim Einfügen über einen „view“ sind alle vorstehenden Restriktionen zu beachten, und es darf keine „NOT NULL“-Bedingung der realen Tabelle verletzt werden, d.h. alle „NOT NULL“-Felder müssen im „view“ enthalten sein.

Mit **`DROP TABLE ...`** / **`DROP VIEW ...`** / **`DROP INDEX ...`** können Tabellen, Sichten, Indexe gelöscht werden, z.B.:

```
DROP VIEW EINKOMMEN;
DROP TABLE ABTKOSTEN;

DROP INDEX ABT_IDX;
DROP INDEX JOB_IDX;
DROP INDEX ANGEST_IDX;
DROP INDEX QUAL_IDX;

DROP TABLE QUALIFIKATION;
DROP TABLE ANGESTELLTE;
DROP TABLE JOB;
DROP TABLE ABTEILUNG;
```

3. Ändern der Tabellenstruktur

Zum Einfügen neuer Felder dient der `ALTER TABLE` ... –Befehl mit der `ADD`-Klausel, z.B.:

```
ALTER TABLE ABTKOSTEN ADD(ANZAHL INTEGER);
```

Reicht der zur Verfügung gestellte Platz von Tabellenspalten nicht aus, dann kann über `ALTER TABLE` ... mit einer `MODIFY`-Klausel die Bereichsgröße dieser Felder erweitert werden.

4. Hinzufügen von Datensätzen (Zeilen)

Zum Einfügen eines einzelnen neuen Datensatzes dient der `INSERT`-Befehl mit einer `VALUE`-Klausel, z.B.:

```
insert into abteilung values
 ('KO','Konstruktion');
.....................
insert into job values
('KA','Kaufm. Angestellter',3000.00);
.....................................
insert into angestellte values
 ('A1','Fritz','02-JAN-50','OD','SY');
.....................................
insert into qualifikation values
```

```
('A1','SY');
.....................................
```

5. Aktualisieren von Daten

Beim Ändern von Sätzen werden die zu verändernden Felder angegeben und die zu ändernden Sätze durch eine Selektion bestimmt, z.B.:

```
update abtkosten
set anzahl =
 (select count(angestellte.abt_id)
 from abtkosten, angestellte
 where abtkosten.abt_id = 'KO' and
       angestellte.abt_id = 'KO'
 group by abtkosten.abt_id)
where abtkosten.abt_id = 'KO';
```

Falls die WHERE-Klausel fehlt, werden alle Spalten nach der Angabe in der SET-Anweisung verändert.

6. Löschen von Daten

Das Löschen von Datensätzen kann durch eine Selektion spezifiziert werden:

```
DELETE
FROM <Tabellen-Name>
WHERE <Bedingung>
```

Weitere Klauseln des SELECT-Befehls sind in der DELETE-Anweisung nicht sinnvoll und daher auch nicht zugelassen. Ohne WHERE-Klausel wird der Inhalt det Tabelle gelöscht, z.B.:

```
DELETE * FROM ANGESTELLTE;
```

7. Zusammenfassung

```
ALTER TABLE <TABLE NAME> ADD|DROP|MODIFY (<Spalten
Spezifikationen...>[50]);
```

`COMMIT;`
Macht alle Veränderungen seit der letzten Transaktion permanent und schließt die aktuelle Transaktion ab. Die mit `INSERT`, `DELETE` und `UPDATE` gesetzten Änderungen werden nicht sofort der Datenbank übergeben. Der Benutzer sieht zwar eine konsistente Sicht seiner Änderungen, andere Benutzer sehen diese Änderungen aber noch nicht. Erst wenn der Befehl `COMMIT` gegeben wird, gehen die gesammelten Änderungen in die Datenbank ein und werden öffentlich.

```
CREATE [UNIQUE] INDEX <Indextabellen-Name>
ON <TABLE NAME> (<SPALTEN LISTE>);[51]
```

[50] Vgl. CREATE-Befehl

```
CREATE TABLE <Tabellen-Name>
(<Spalten-Name> <Datentyp> [(<Größenangabe>[52])]
<Spaltenbedingung>[53],...<weitere Spalten>);[54]

CREATE TABLE <Tabellen-Name>
[(Spaltendefinition1, ... , Spalten-definitionN[55])]
AS
<select-Anweisung>[56];

CREATE VIEW <Tabellen-Name> AS <Abfrage>;

DELETE FROM <Tabellen-Name> [WHERE <Bedingung>];

INSERT INTO <Tabellen-Name> [(<Spalten-Liste>)]
VALUES (<Wert-Liste>);

ROLLBACK;
```

Macht alle Veränderungen an der Datenbank seit dem letzten COMMIT-Kommando wieder rückgängig.

```
SELECT [DISTINCT|ALL] <Spalten-Liste, Funktionen,
                       Konstanten, etc.>
FROM <Tabellennamen-Liste oder Namen von Views>
[WHERE <Bedingung(en)>]
[GROUP BY <Gruppierte Spalte(n)>]
[HAVING <Bedingung>]
[ORDER BY <Geordnet nach Spaltenwerte> [ASC|DESC]];

UPDATE <Spalte(n)>
SET <Spalten-Name> = <Wert>
[WHERE <Bedingung>];
```

SQL arbeitet mit verschiedenen Objekten (Tabelle, Spalte, Zeile, View, skalare Werte) und Operationen auf diesen Objekten, wie die verschiedenen Syntax-Diagramme zum SELECT-Kommando zeigen. SQL besteht aber nicht nur aus der SELECT-Anweisung, sondern setzt sich insgesamt aus 12 Befehlen zusammen. Diese Befehle lassen sich zur Datendefinitions-Sprache (Data Definition Language, **DDL**), der Datenmanipulationssprache (Data Manipulation Language, **DML**) und der Datenkontroll-Spache (Data Control Language, **DCL**) zuordnen. Außerdem gehört zu SQL der **Systemkatalog** zur Verwaltung der Tabellenstruktur und Werkzeuge für "Performance"-Optimierung zur Beschleunigung der Arbeits-abläufe.
Die **DDL** gibt die Definition von Tabellen-, View- und Indexdefinitionen an und benutzt dazu die Befehle: CREATE, DROP, ALTER

---

[51] UNIQUE steht in eckigen Klammern und ist deshalb eine optionale Angabe
[52] Die Größenangabe ist nur bei bestimmten Datentypen erforderlich
[53] NULL bzw. NOT NULL und UNIQUE.
[54] Die Angaben zu den Spalten sind auch in Verbindung mit ALTER TABLE ... möglich.
[55] Namen der Spalten, die mit den von der Unterabfrage zurückgegebenen Werten assoziiert werden sollen
[56] Zulässige „select“-Anweisung, die beim Erzeugen der neuen Tabelle verwendet wird.

Die **DML** vollzieht das Suchen, Einfügen, Aktualisieren und Löschen und benutzt dazu die Befehle: `SELECT`, `INSERT`, `UPDATE` und `DELETE`.
Die **DCL** umfaßt drei verschiedene Arbeitsgebiete:

-Recovery und Concurrency (Transaktionen und Regeln für die Verfahrensweise bei Mehrfachzugriffen) mit den Befehlen `COMMIT`, `ROLLBACK`, `LOCK`
- Sicherheit (bzgl. der Zugriffsrechte) mit den Befehlen `GRANT`, `REVOKE`
- Integrität (Einschränkungen für den Erhalt der Korrektheit von Daten)

In gewisser Hinsicht ist `SQL` gegenwärtig die standardisierte Datenbanksprache. Es bestehen jedoch (- wie in fast jedem Standard -) Erweiterungen der jeweiligen SQL-Hersteller, die eine vollständige Kompatibilität nicht möglich machen. Alle sinnvollen Erweiterungen werden jedoch von den Normungsgremien ISO[57] und ANSI für eine umfangreiche Sprachdefinition gesammelt. 1992 wurde eine wesentlich erweiterte Fassung der SQL-Norm (SQL2) [58] veröffentlicht. Sei längerer Zeit wird bei ISO (parallel zu SQL2) am Projekt SQL3 gearbeitet und folgende Erweiterungen zu SQL2 diskutiert:

- Unterstützung komplexer Datenstrukturen
- Sprachmittel (Ausdrucksmöglichkeiten) für Datenbankprozeduren
- objektorientierte Konzepte, z. B. abstrakte Datentypen
- Unterstützung verteilter Datenbanken

SQL ist Bestandteil zahlreicher Datenbanksysteme (z.B. dBASE IV, Oracle, Database Manager[59] bzw. IBM Database 2 (DB 2).

[57] ISO 9075: 1987: Database Language SQL 1987 bzw. ISO 9075: 1989: Database Language SQL with Integrity Enhancement 1989
[58] ISO 9075: 1992: Database Language SQL, 1992
[59] Es handelt sich hierbei um ein Datenbanksystem, das unter OS/2 läuft und wird im Rahmen von OS/2-Auslieferungen der IBM bereitgestellt

1.5 Klassifikation der DB-Anwendungen

1.5.1 Elementare Anwendungsformen

Stapelverarbeitung

DB-Anwendungen im Stapelbetrieb werden heute noch dann priorisiert, wenn ein hoher Durchsatz der DB-Programme angestrebt wird

Dialogverarbeitung

Sie steht unter der Prämisse, mit möglichst geringer Antwortzeit und ohne Programmieraufwand auf jede **DB** zugreifen zu können. Eine Tendenz zum Ausbau der Online-Anwendung vom reinen Abfrage- und Erfassungsbetrieb zu Online-Update (Abfrage, Erfassen, Ändern, Löschen) oder gar Realzeitverarbeitung ist unverkennbar.

Interaktive Verarbeitung datenorientierter Aufgabenstellungen

Datenbanken lösen datenorientierte Aufgabenstellungen in interaktiver Verarbeitung. Ein derartiges Anwendungssystem kann in drei Basiskomponenten gegliedert werden:

- eine Präsentationskomponente zur Realisierung der Benutzerschnittstelle
- eine Logikkomponente zur Ausführung von Verarbeitungsfunktionen und Über-nahme des Kontrollflusses in Anwendungssystemen
- eine Datenkomponente mit der Aufgabe der Datenverwaltung

Die Datenkomponente kann weiter (sog. **5-Schichten-Architektur**)[60] unterteilt werden in:

- das Datenbanksystem
  Das ist heute in der Regel ein relationales Datenbanksystem. Anforderungen an das System werden in SQL formuliert. Das System löst mengenorientierte Anforderungen in satzorientierte Zugriffe (Sätze von Tabellen) auf. Im Rahmen der 5-Schichen-Architektur spricht man hier von der mengenorientierten Schnittstelle. Diese Schnittstelle umfaßt eine Klasse von Hochsprachen, z.B. SQL.
- das Datensatzverwaltungssystem (Recordmanagementsystem)
  Es löst Anforderungen nach Datensätzen über die Kenntnis von Zugriffspfaden (z.B. über einen Index) auf und bestimmt die Position der angeforderten Sätze in den Dateien (Tabellen). Das Datensatzverwaltungssystem setzt Aufrufe an das Dateisystem ab. Im Rahmen der 5-Schichten-Architektur spricht man von der satzorientierten Schnittstelle. Der Zugriff zur Datenbasis erfolgt satzweise.
- Pufferverwaltung, Einbringstrategie
  Von der satzorientierten Schnittstelle aus will man die Abspeicherung von Satzmengen steuern. Zur Kontrolle der Transportvorgänge von und zum Hinter-grundspeicher ist ein homogener und linearer Speicher wünschenswert, der Einzelheiten (z.B. Dateiorganisation, Pufferung, Blockgruppenanordnung) verdeckt. Homogen und linear bedeutet: Unterteilung in Adressierungseinheiten fester Größe (sog. Seiten), auf die direkt zugegriffen werden kann. Der Seitenorganisation wird häufig noch eine Segmentstruktur[61] überlagert, mit der sich die Zusammengehörigkeit der Daten ausdrücken läßt. Aufgabe dieser Schicht ist die Umsetzung der Segmente und Seiten auf Dateien und physische Blöcke

[60] vgl. 3.1.1

[61] Man spricht auch von der Segmentschnittstelle

(Einbringstrategie). Eine Pufferverwaltung sorgt dafür, daß benötige Seiten zur Verfügung stehen, ohne daß der Benutzer sich um Seitentransport oder Strategien zum Seitenaustausch kümmern muß.

- das Dateisystem
  Es stellt höheren Schichten Operationen zum Lesen, Einfügen, Ändern und Löschen von Bereichen in Dateien (Tabellen) zur Verfügung. Weitere Aufgaben sind: Das Sperren von Bereichen, die Manipulation von Dateien und Dateiverzeichnissen. Zur Realisierung dieser Aufgaben bedient sich das Dateisystem der tieferen Schicht eines Betriessystems.
- das System der physikalischen Ein-, Ausgabe
  Diese Komponente realisiert den Zugriff auf die Speicherperepherie. Diese Schicht greift über die Zugriffsmethode unmittelbar auf den Speicher zu und steuert die Übertragung von bzw. zum Systempuffer. Im Rahmen der 5-Schichten-Architektur spricht man hier von der Geräteschnittstelle (mit den diversen Treiberprogrammen).
  Damit werden Daten, die sich bspw. dem Benutzer im Relationenmodell noch als Tupel darstellen, vom Datenbankverwaltungssystem auf Seiten (d.s. Blöcke fester Länge in der Größenordnung von 512 Bytes bis 8 KBytes) abgebildet. Mengen derartiger Seiten sind jeweils zu linearen Adreßräumen zusammengefaßt, die Segmente[62] genannt werden. In den Seiten sind Records (Sätze) fester oder variabler Länge abgelegt. Solche Records können Tupel oder auch Verweislisten in einem Zugriffspfad sein. Recordoperationen werden demnach auf lesende und schreibende Seitenzugriffe abgebildet.

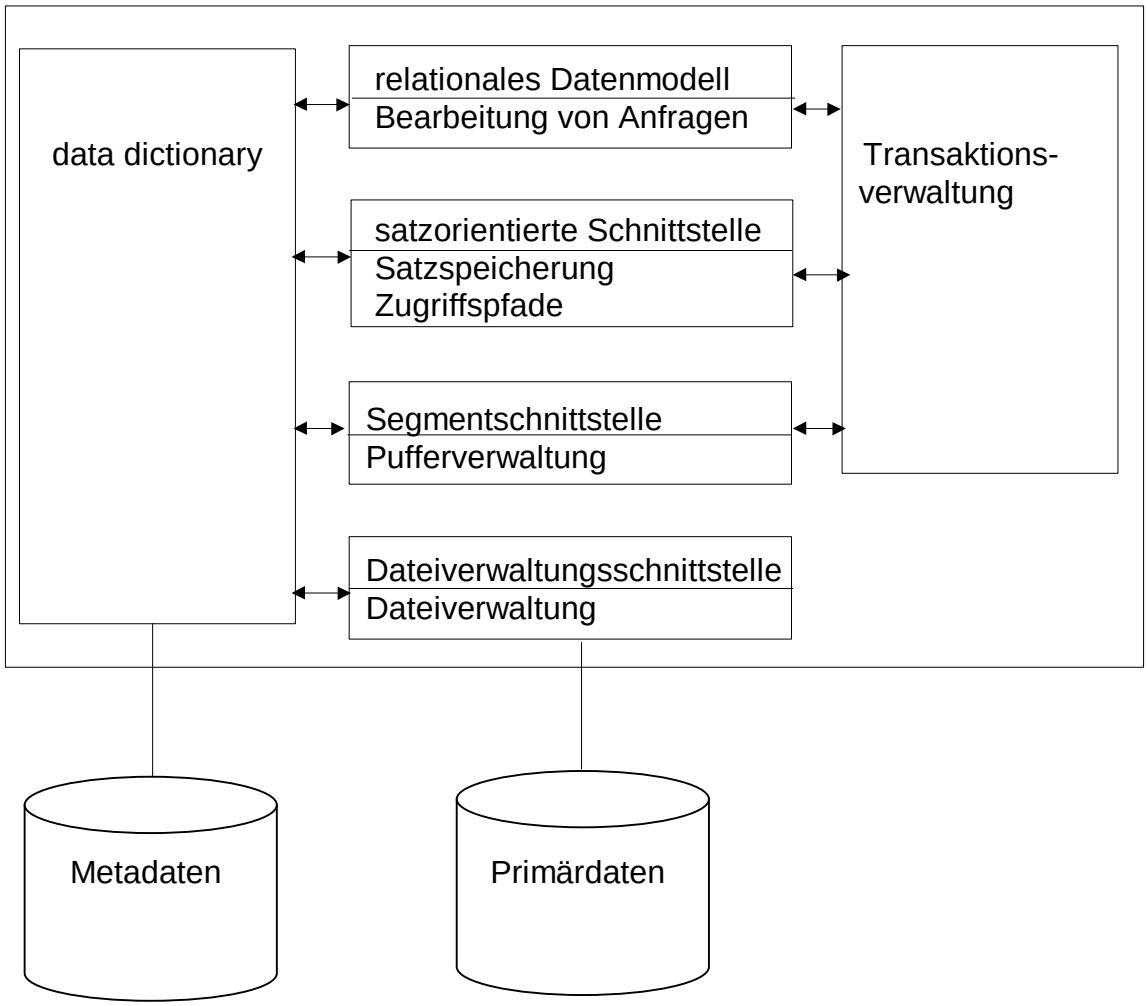


Abb. 1.5-1: Architektur von Datenbanksystemen

[62] Synonyme: Realm, Area, DB Space

1.5.2 Transaktionsbetrieb

1.5.2.1 Transaktionen

Definition und Erklärungen

Bei größeren Datenbeständen ist es unumgänglich, daß sie von mehreren Benutzern gleichzeitig bearbeitet werden. So können häufig wiederkehrende Geschäftsvorgänge sofort durch Zugriff von Datenstationen über ein Anwendungs-programm auf zentrale Datenbestände erledigt werden.
Eine (DC-) **Transaktion** (TA) ist als Folge von (Dialog-) Datenbank- und Verarbeitungsanweisungen definiert, die einen konsistenten Datenbestand in einen wieder konsistenten Bestand überführen. **Konsistenz** bedeutet: Der Datenbestand ist logisch widerspruchsfrei.
Eine Transaktion umfaßt demnach eine oder mehrere DB-Operationen, für die die DB-Software sog. Transaktionseigenschaften gewährleistet. Dazu zählt bspw. die "Alles-oder-nichts"-Eigenschaft: Änderungen einer Transaktion werden vollständig oder überhaupt nicht in die Datenbank übernommen. Erfolgreiche Transaktionen sind dauerhaft: Sie gehen trotz möglicher Fehlersituationen (Rechner-, Plattenausfall, Fehler im Kommunikationsnetzwerk) nicht mehr verloren. Dafür sorgen spezielle "Log-Dateien", die alle Änderungen protokollieren. Das DBMS sorgt bei Ausfällen für geeignete Recovery-Maßnahmen.

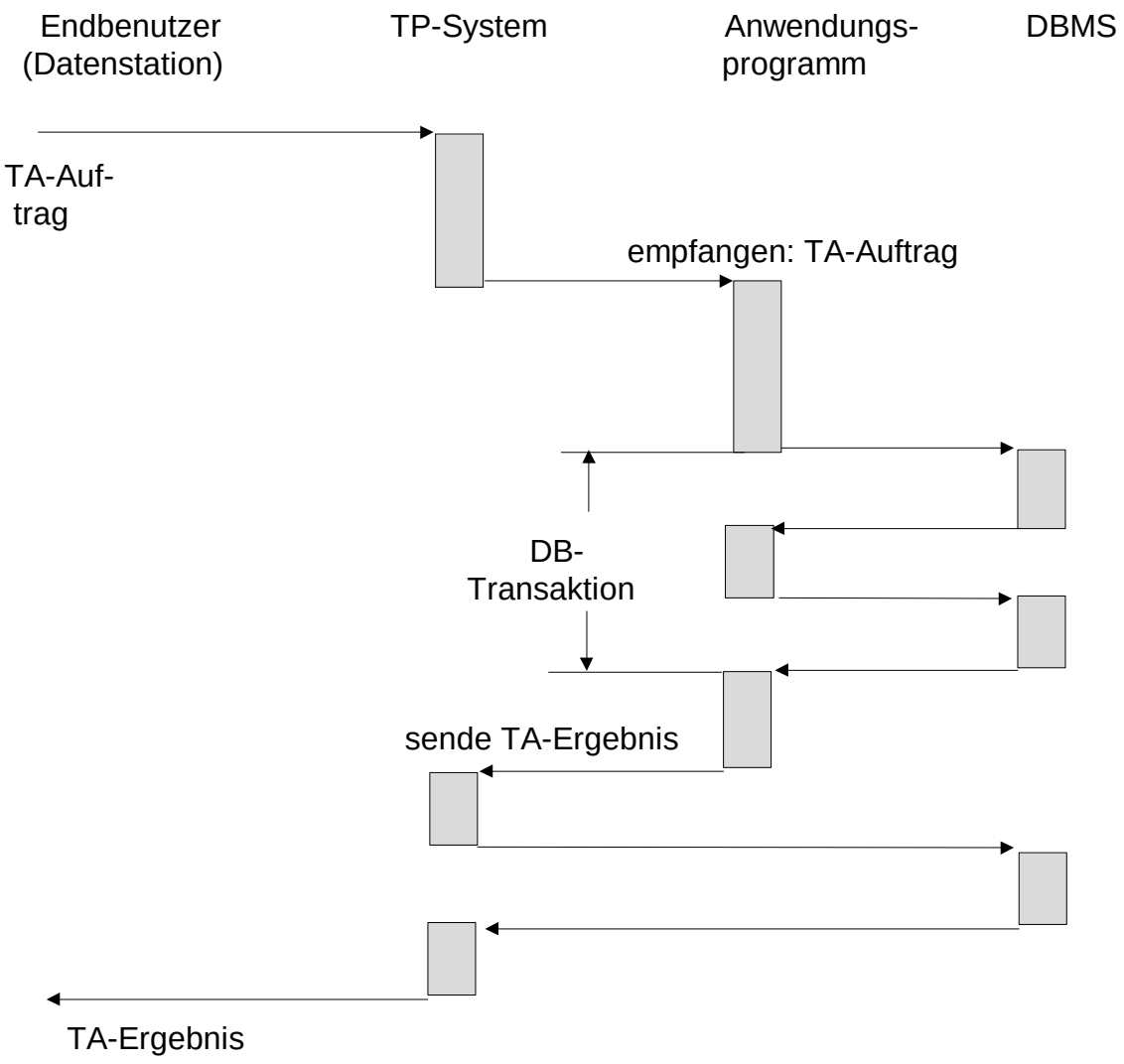


Abb. 1.5-2: Bearbeitung eines Transaktionsauftrages

Transaktionsparallelität

Im Mehrbenutzerbetrieb befinden sich in der Regel gleichzeitig mehrere Transaktionen in Bearbeitung. Von einer häufig sogar großen Anzahl von Endgeräten kommen Anforderungen (requests) , z.B. Auskunftsersuchen, Buchungsaufträge, Einzahlungen, Bestellungen, Rechnungen. Jedem Auftrag ist ein Anwendungsprogramm zugeordnet. Es nimmt Anforderungen vom Endgerät entgegen, führt die notwendigen Verarbeitungsschritte einschl. der Datenbankzugriffe aus und meldet das Ergebnis an das Endgerät zurück. Jede Instanziierung eines derartigen Anwenderprogramms wird als Transaktion ausgeführt. Das bedeutet: Die Ausführung einer Transaktion ist bzgl der Datenbank und bzgl. des Nachrichtensystems ununterbrechbar. Ein Synchronisationsmechanismus sorgt dafür, daß alle Programme sich so verhalten, als würden sie seriell (d.h. in Einbenutzer-Umgebung) ausgeführt.
Transaktionsparallelität entsteht durch „time sharing“. Die Anwendung bestimmt lediglich, welche und wieviel Arbeitsschritte zu einer Transaktion zusammengefaßt werden. Paralleltät zwischen Transaktionen steigert den Durchsatz durch Erhöhung der Zahl der Verarbeitungsprozesse. Ein Übermaß an „time sharing“ verlängert die Antwortzeit der einzelnen Transaktionen. Zu viele gleichzeitig aktive Transaktionen bewirken ein Übermaß an Synchronisierungskonflikten. Die Betriebssystemumgebung legt fest, wie viele Transaktionen bearbeitet werden sollen. Das DBS muß nur dafür sorgen, daß die **ACID**-Eigenschaft der Transaktionen (**A**tomicity, **C**onsistency, **I**solation, **D**urability) erhalten bleiben.

Die ACID-Eigenschaft

Transaktionen müssen die ACID Eigenschaften erfüllen:

- Atomarität (atomicity)
- Konsistenz (consistency)
- Isolierte Zurücksetzbarkeit (isolation)
- Dauerhaftigkeit (durability)

Unter **Atomarität** wird verstanden, dass alle zu einer Transaktion gehörenden Aktionen als eine logische Einheit angesehen werden und nur komplett ausgeführt werden dürfen ("Alles-oder-nichts"-Eigenschaft). Schlägt eine Aktion fehl, müssen alle bereits vorgenommenen Veränderungen rückgängig gemacht werden, die Transaktion wird zurückgesetzt.
Mit der **Konsistenz**-Eigenschaft wird gefordert, dass sich die Datenbank nach Ablauf der Transaktion in einen Zustand befindet, der die Konsistenz[63] bzw. Integrität[64] nicht verletzt. Kann dieser Zustand nicht erreicht werden, so muss zum Ausgangszustand zurückgekehrt werden.
Isolierte **Zurücksetzbarkeit** bedeutet, dass durch das Zurücksetzen einer Transaktion keine andere Transaktion so beeinflusst, dass diese ebenfalls zurückgesetzt werden muss. Die Transaktion muss alle Zugriffe auf gemeinsam

[63] Konsistenz: Der Datenbestand ist logisch wiederspruchsfrei.
[64] Integrität allg.: Übereinstimmung von realen und gespeicherten Daten.

genutzte Ressourcen serialisieren und garantieren, dass sich die konkurrierenden Programme nicht beeinflussen. Im Mehrbenutzerbetrieb mit vielen parallelen und überlappenden Transaktionen muss sich ein Programm also genauso verhalten wie im Einbenutzerbetrieb.

Unter **Dauerhaftigkeit**[65] versteht man, dass von einer erfolgreich abgeschlossenen Transaktion vorgenommenen Änderungen auch tatsächlich in der Datenbank wirksam werden und nicht mehr (z.B. in Folge eines Systemzusammenbruchs) einfach verloren gehen können.

Das Zustandsdiagramm einer Transaktion

Jeder im Mehrbenutzerbetrieb arbeitende Transaktion kann genau ein Zustand zugeordnet werden, in dem sie sich gerade befindet:

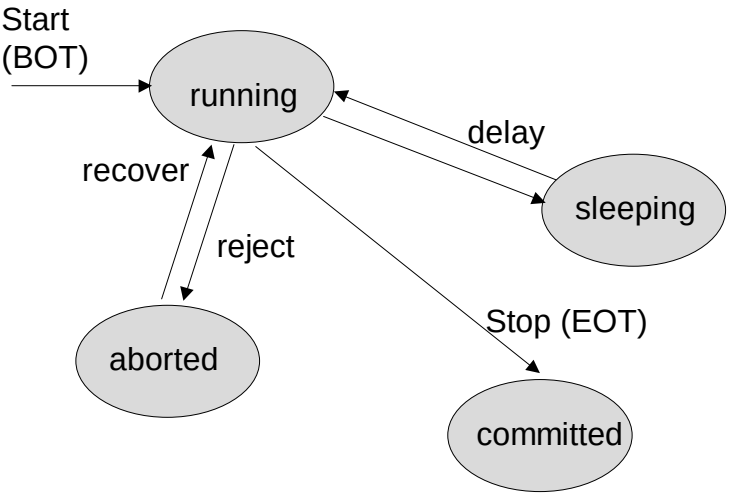


Abb. 1.5-3: Zustandsdiagramm einer Transaktion

Aktionen (logische Verarbeitungseinheiten) einer Transaktion sind durch BOT (Begin of Transaction) bzw. EOT (End of Transaction) begrenzt. Nach dem Start befindet sich eine Transaktion im Zustand "running". Hier wird ihr das Betriebsmittel Datenbank für eine bestimmte Zeit zur Verfügung gestellt. Ist der Auftrag danach abgewickelt, dann geht sie in den Zustand "committed" über, falls die EOT-Marke erreicht ist. Ist der Auftrag nicht vollständig abgewickelt, dann geht die Transaktion in den Zustand "sleeping" über. Die Bearbeitung wird zu einem späteren Zeitpunkt fortgesetzt. Kommt es während der Bearbeitung der Transaktion zu Konsistenzverletzungen, dann wird die Transaktion zurückgewiesen und geht in den Zustand "aborted" über. In diesem Fall wird zu einem späteren Zeitpunkt vollständig neu gestartet, alle Änderungen an der Datenbank werden rückgängig gemacht.

Fehler im Transaktionsbetrieb

Fehlerursachen können bspw. sein

a) Programmfehler

[65] Persistenz ist ein Synonym für Dauerhaftigkeit

b) unerlaubt hoher Verbrauch von Ressourcen
c) Verlust von Hauptspeicherinhalten
d) Betriebssystemabsturz
e) Zusammenbruch des Datenbanksystems
f) Plattenfehler (im schlimmsten Fall sog. Head-Crash)

Nur die Fehlertypen a) bis b) sind Transaktionsprogrammen unmittelbar anzulasten und können im laufenden Betrieb durch die „**Recovery**"-Komponente des Datenbanksystems repariert werden.
Fehler vom Typ c) bis e) heißen „Soft"-Crash. Maßnahmen zur Wiederherstellung der Datenbank nach solchen Fehlern werden unter dem Begriff „Crash Recovery" zusammengefaßt. Sie erfordern einen Wiederanlauf des Datenbanksystems.
Nach einem Fehler vom Typ f) ist i. a. die aufwendige „Archiv-Recovery" einzuleiten.

Anomalien im Mehrbenutzerbetrieb

Ein Reihe von Fehlern (Anomalien) kann im Mehrbenutzerbetrieb auftreten, selbst wenn jede einzelne Transaktion für sich allein feherfrei abläuft und das System reibungslos funktioniert. Derartige Anomalien können sein:

„lost update"
Die Ausführung unterschiedlicher Transaktionen auf eine gemeinsame Datenbank kann zu Problemen führen (z.B. „lost update", „out-of-date retrieval"). So bewirken die folgenden Arbeitsschritte der Transaktionen T_1 und T_2, die nebeneinander (nebenläufig) ablaufen, den Verlust („lost update") der durch die Transaktion T_1 durchgeführten Aktualisierung:

1. Transaktion T_1 holt das Objekt aus der Datenbank
2. Transaktion T_2 liest das Objekt aus der Datenbank
3. Transaktion T_1 aktualisiert das Objekt und schreibt es zurück
4. Transaktion T_2 aktualisiert das Objekt und schreibt es zurück in die Datenbank

„out-of-date retrieval"
Zu einer veralteten Wiedergewinnung („out-of-date retrieval") führt dagegen die folgende Reihenfolge der Ereignisse:

1. Transaktion T_1 liest das Objekt zur Aktualisierung aus der Datenbank
2. Transaktion T_2 liest das Objekt zur Information
3. Transaktion T_1 verändert das Objekt und schreibt es zurück in die Datenbank

Lese-/Schreibsperren

Hier müssen für die Dauer der Ausführung einer Transaktion die betroffenen Daten (sog. Konsistenzbereich) gegen den Zugriff anderer Anwender geschützt werden. So muß eine der beiden Transaktionen den Zugriff bzgl. des Objekts sperren. Die Wahl bzw. die Definition des Konsistenzbereichs soll derart geschehen, daß nur der möglichst kleinste definierbare, aber notwendige Bereich gesperrt wird.

Eine **Lesesperre** erlaubt den Lesezugriff auf das Objekt und verhindert, daß eine Transaktion das Objekt aktualisiert. Eine beliebige Anzahl Transaktionen kann eine Lesesperre auf ein Objekt zur gleichen Zeit ausüben.
Ein **Schreibsperre** erlaubt Lese- und Schreibzugriff auf ein Objekt, hindert andere Transaktionen am Lesen und Schreiben des Objekts. Eine Transaktion erhält exklusiven Zugriff auf das Objekt.
Die Größe der Dateneinheiten, die in einem Datenbanksystem gesperrt werden können, wird als **Granularität** (degree of granularity) bezeichnet. In relationalen Datenbanksystemen sind typischerweise folgende sperrbare Einheiten bekannt:

- die Datenbank
- die Relation
- ein physischer Plattenblock
- ein Attributwert

Verklemmungen

Je nach Art der verwendeten Sperren entstehen zwischen Transaktionen Wartebeziehungen bzw. Serialisierungsbedingungen durch Zugriffe auf die Datenbank.
Falls Transaktionen Datenelemente sperren, kann es zu Verklemmungen kommen, z.B.: Die Transaktion T_1 und T_2 können nebenläufig ausgeführt werden. Das geschieht zufällig in folgender Reihenfolge:

1. Transaktion T_1 sperrt Objekt A zum Schreiben
2. Transaktion T_2 sperrt Objekt B zum Schreiben
3. Transaktion T_1 fordert eine Sperre auf Objekt B, muß aber warten, da Objekt B von T_2 gesperrt ist.
4. Transaktion T_2 fordert eine Sperre auf Objekt A, muß aber warten, da Objekt A von T_1 gesperrt ist.

An diesem Punkt können weder T_1 noch T_2 fortfahren. Sie sind verklemmt.

Synchronisation von Transaktionen (TA)

Es gehört zur Aufgabe der Transaktionsverwaltung, derartige unkorrekten Abläufe (Anomalien) voneinander zu isolieren und zu synchronisieren.
Die Abarbeitung einer Transaktion läßt sich durch das **Zwei-Phasen-Protokolls** ohne Gefährdung des Datenbestands und ohne Verklemmungen steuern und kontrollieren:

- Phase 1
Sperrphase des Konsistenzbereichs der Transaktion
Bevor eine Transaktion auf ein Objekt liest oder schreibt, muß sie das Objekt sperren
- Phase 2
Bearbeitung der Daten und Freigabe (auch sukzessiv) des Konsistenzbereichs

Hat eine Transaktion einmal eine Sperre wieder freigegeben, darf sie keine weiteren Sperren mehr anfordern bzw. erhalten, d.h.: In einer Transaktion sind alle Sperren vor allen Freigaben angeordnet.

Der Anwendungsprogrammierer soll sich aber nicht um derartige Maßnahmen zur Vermeidung von Anomalien kümmern müssen. Dieses Ziel kann erreicht werden, indem jede Transaktion automatisch so gesteuerrt wird, daß er scheinbar alleine auf der Datenbank arbeitet. Die sog. **Concurreny-Control**-Komponente der Transaktionsverwaltung muß die Äquivalenz der parallelen Ausführung einer Menge von Transaktionen zu einer seriellen, d.h. nicht überlappenden Reihenfolge gewährleisten. Eine parallele Ausführung mehrerer Transaktionen mit dieser Eigenschaft heißt serialisierbar.

1.5.2.2 Das Zwei-Phasen-Commit-Protokoll

Dieses Protokoll wird verwendet um lokale Transaktionen an verschiedenen Programmen oder Maschinen (die Subtransaktionen) zu synchronisieren, so dass **entweder alle** erfolgreich durchgeführt werden **oder keines.**
Dabei nehmen die Subtransaktionen vor ihrem eigentlichen Commit einen Zwischenzustand, den ***Ready-to-Commit-Zustand***, ein. In diesem Zustand garantieren die Subtransaktionen, dass das Commit, falls vom Koordinator gewünscht, garantiert ausgeführt wird (selbst wenn zwischendurch ein Systemabsturz stattfindet) und dass aber auch noch ein Abbruch der Gesamttransaktion akzeptiert wird und dann die Subtransaktion zurückgesetzt wird, sog. rollback[66].
Ablauf des 2PC-Protokolls: Alle beteiligten Transaktionen werden durch eine „Prepare-to-Commit"-Anweisung des Commit-Managers aufgefordert, den Ready-to-Commit-Zustand einzunehmen. Die Subtransaktionen können wieder verteilte Transaktionen abgesetzt haben, zu welchen sie das Prepare-to-Commit Kommando weiterleiten müssen. So kann ein Transaktions-Baum entstehen mit dem Commit Manager als Wurzel.
Der Manager wartet dann auf die Bestätigung aller beteiligten Stationen, dass sie den Ready-to-Commit-Zustand eingenommen haben. Ist diese Bestätigung eingetroffen, wird sie in einem sicheren Platz zwischengespeichert, damit sich das System auch nach einem Wurzelknotenausfall in den Ready-to-Commit-Zustand zurückversetzen kann. Damit ist die erste Phase des 2PC Protokoll abgeschlossen.
Ein Commit wird vom Manager an alle Knoten gesendet. Haben die Subtransaktionen wiederum Subtransaktionen, so muss auch an diese die Commit- Aufforderung weitergegeben werden usw.. Auf diese Weise wird der gesamte Transaktionsbaum durchlaufen.
Die zweite Phase des 2PC- Protokolls ist beendet, wenn alle beteiligten Knoten eine Commit-Bestätigung zum Manager gesendet haben. Jetzt kann dem Client der erfolgreiche Abschluss der Transaktion mitgeteilt werden.
Die Transaktion wird abgebrochen, wenn eine Subtransaktion fehlschlägt und die Commit-Aufforderung zurückweist. Dann fordert der Manager alle Subtransaktionen (und diese wiederum ihre Subtransaktionen usw.) auf, ein Rollback durchzuführen. D.h. alle beteiligten Stationen machen die getroffenen Veränderungen rückgängig und versetzen sich wieder in den Ausgangszustand.

[66] Rollback: Zurücksetzen des Systems in den Ausgangszustand vor der Transaktion (z.B. werden Veränderungen der Daten in der DB rückgängig gemacht).

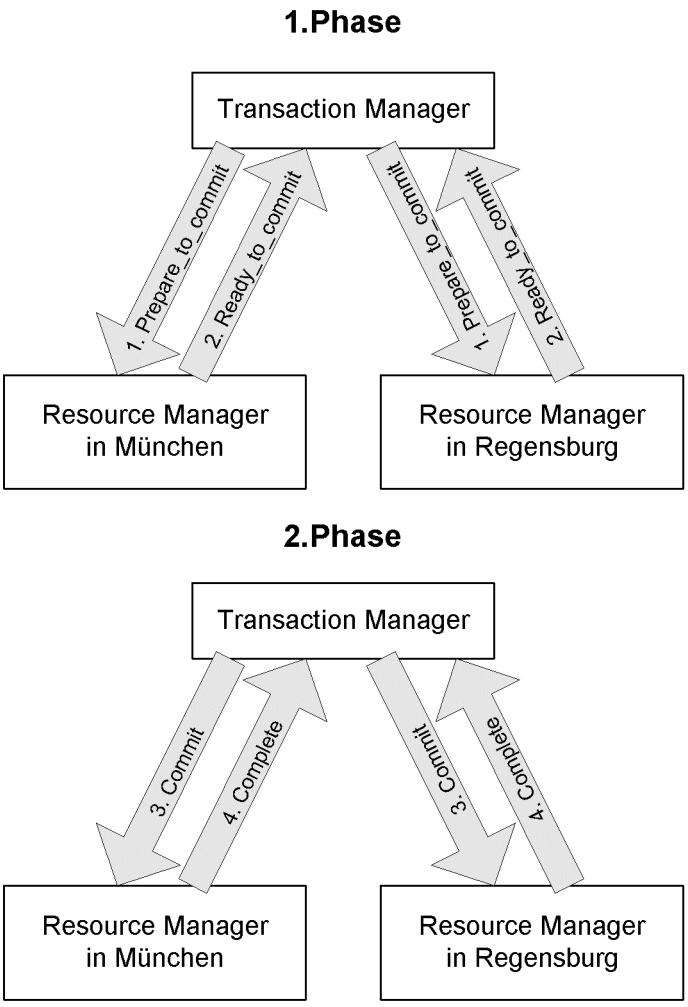


Abb.: Ablauf des 2PC-Protokoll

Der größte Nachteil des 2PC Protokoll ergibt sich aus der hohen Anzahl der Nachrichten die versendet werden müssen. Dies führt zu Performanz-Einbußen. Nicht jede Transaktion braucht die Sicherheit eines 2PC Protokolls (z.B. read-only Transaktionen). Diese müssen erkannt und durch ein einfaches Protokoll abgearbeitet werden.

1.5.2.3 Transaktionsmonitor

Operating Systeme für Transaction Processing

Den Transaktionsbetrieb mit Datenbankanwendungen realisieren **DC**-Komponenten. Man spricht auch von einem „**Transaktionssystem** (TP-System, transaction processing system, **Transaktionsmonitor**)“. Dieses Systemprogramm koordiniert für viele Endbenutzer die unterschiedlichen Transaktionsaufträge und unterstützt dabei die logisch zusammenhängenden Verarbeitungssequenzen (Transaktionen).
Bei der Implementierung eines Transaktionssytems muß berücksichtigt werden: Eine (meist) große Anzahl von Benutzern ruft gleichzeitig Funktionen mit Zugriffen auf die gemeinsamen Datenbestände ab. Die Koordinierung übernimmt ein **Transaktionsmonitor** (TP-Monitor), der den Anwendungsprogrammen alle Fragen der Mehrfachausführung und der gerätespezifischen Ein-/Ausgabe abnimmt. Transaktionsprogramme greifen natürlich auf die Datenbank zu. **TP-Monitor** und Datenbanksystem müssen daher gleichzeitig aktiv sein und koordiniert zusammen arbeiten. Man unterscheidet nichtintegrierte DB/DC-Systeme von integrierten DB/DC-Systemen.
Nichtintegrierte DB/DC-Systeme entstehen durch einfache Kopplung der unabhängigen DB- und DC-Teile, die je einen eigenen Steuerteil aufweisen (z.B. UDS/UTM [67]). Die Koordinierung von "recovery" und "restart" ist in diesen Fällen kompliziert.
Bei integrierte DB/DC-Systemen ist DC-Komponente Bestandteil des Datenbanksystems.
Aus der Sicht der Anwender ist ein TP-Monitor ein Programm, das eine Endlosschleife mit einer ACID-Transaktion als Schleifenkörper bearbeitet.

[67] UDS ist ein "Universelles Datenbanksystem der Firma SIEMENS, UTM ist ein "TP"-Monitor der Firma SIEMENS

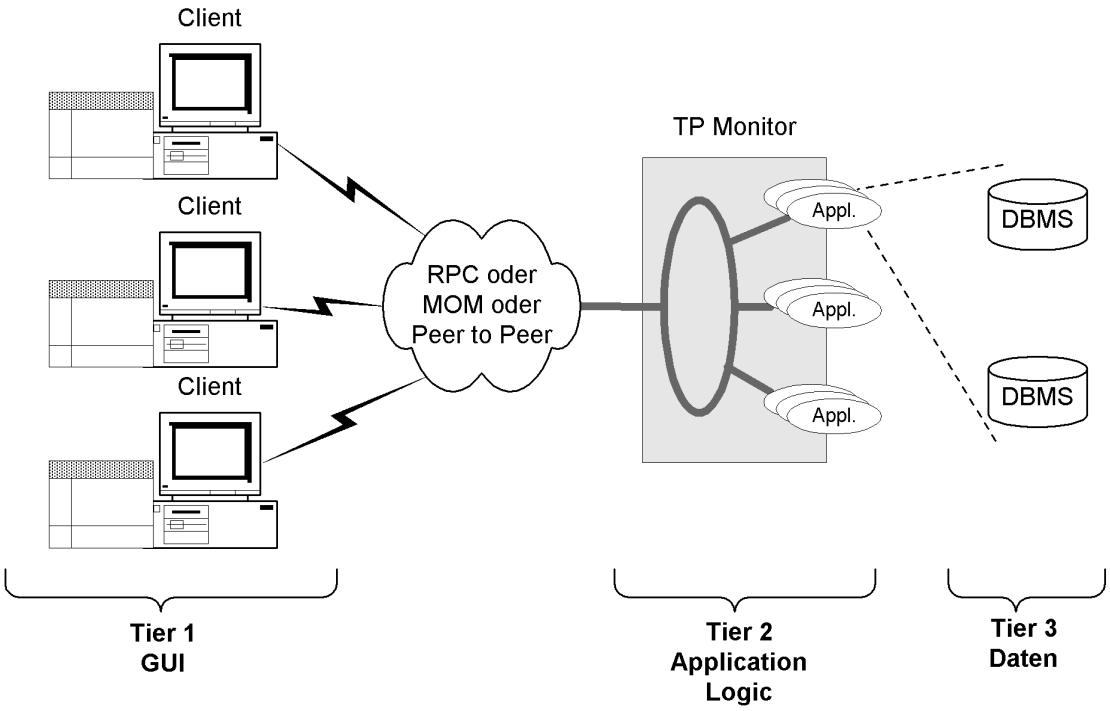


Abb.: Transaktionsmonitor, Beispiel für eine 3-Tier-Anwendung

Der Kontrollfluss innerhalb des TP Monitor für jede Anfrage ist:

1. Den Input vom Display (oder einem anderen Gerät) in das Standard Format für Anfragen (XT API[68]) übersetzen.
2. Auswerten des Anfrage Headers um zu bestimmen, welche Transaktion gefordert wird und welches Programm die Anfrage abarbeiten soll.
3. Starten der Transaktion und des Programms welches in 2. bestimmt wurde (dieses Programm ruft typischerweise Aktionen auf einer Datenbank auf).
4. Commit zum Beenden der Transaktion im Erfolgsfall bzw. Abbruch im Fehlerfall.
5. Zurücksenden eines Output zum Display (oder einem anderen Gerät).

[68] XT API: X/Open Interface Standart für die Kommunikation zwischen Applikation und Transaktion Manager

1.5.3 Client-Server-Systeme

1.5.3.1 Fernzugriff in Netzen aus autonomen Rechnern

Kommunikationsprotokolle

Es ist zweckmäßig, kleinere Datenverarbeitungssysteme zur Führung lokaler Datenbanken für Verarbeitungsaufgaben am Arbeitsplatz vorzusehen und diese über ein Netz an ein oder mehrere Rechenzentren anzuschliessen ("distributed processing"). Jeder Bildschirmarbeitsplatz hat einen eigenen Prozessor und Speicher. Das Anwendungsprogramm läuft auf diesem Arbeitsplatzrechner. Beim Fernzugriff auf die Datenbank findet eine Interprozeßkommunikation zwischen den Anwendungsprogrammen und dem Datenbankverwaltungssystem auf dem Zentralrechner statt.
Das **Client-Server-Modell** ist das dominierende Konzept für **verteilte Datenverarbeitung** mit über ein Kommunikationsnetz verbundenen Rechnern. Man versteht darunter eine Architektur, bei der eine Anwendung in einen benutzernahen Teil (Client, Frontend) und einen von allen Benutzern gemeinsam beanspruchten Teil (Server, Backend) aufgeteilt ist. Voraussetzung der verteilten Datenverarbeitung sind einheitliche Kommunikationsschnittstellen, ein Kommunikationssystem zur Unterstützung der Interprozeßkommunikation, Dienste für die Nutzung der Ressourcen, etc.
Die Kommunikation zwischen Rechnern erfordert Kenntnis und Einhaltung bestimmter Regeln. **Kommunikationsprotokolle** sind Regeln, nach denen Kommunikationspartner Verbindungen aufbauen, Information austauschen und wieder abbauen.
Von der International Standardisation Organisation (ISO) zusammen mit Open System Interconnection (OSI) wurde ein Architekturmodell entwickelt, das heute generell als Ausgangspunkt aller Kommunikationsprotokolle zwischen Rechnern verwendet wird. In diesem Modell werden die einzelnen für die Kommunikation zwischen Rechnern benötigte Dienste in 7 aufeinanderfolgenden Schichten abgewickelt. Jede Schicht erfüllt bestimmte Dienstleistungen bzw. Aufgaben und stellt das Ergebnis der nächsten Schicht zur Verfügung.

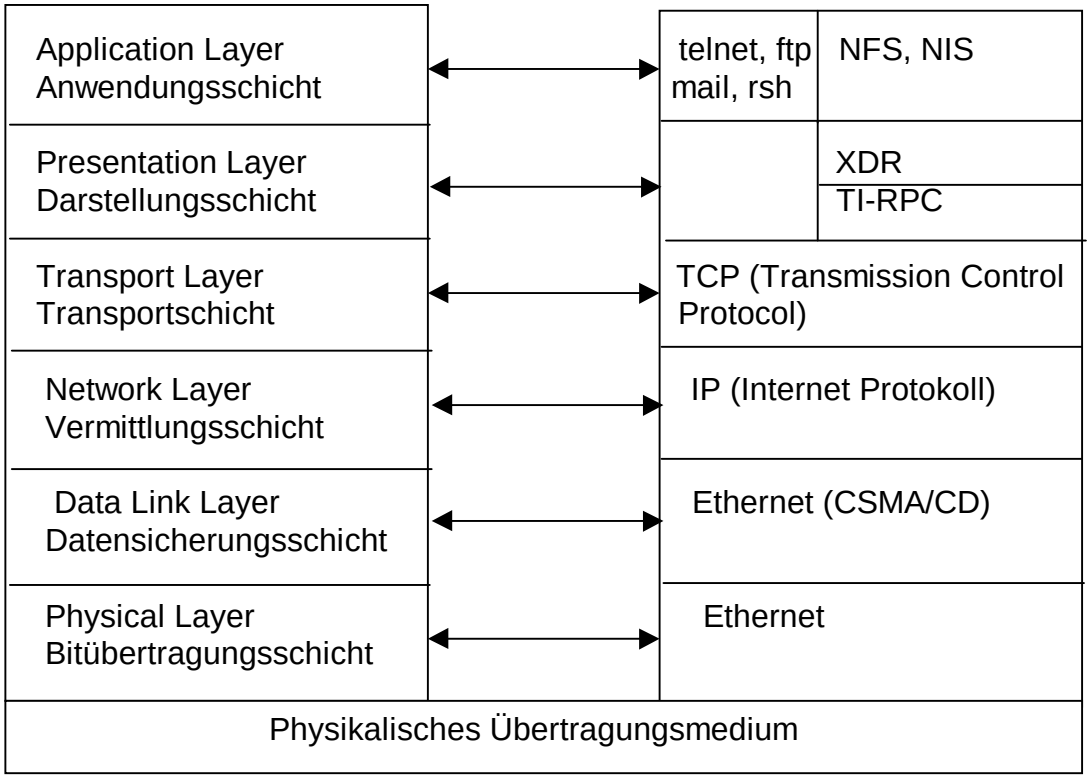


Abb. 1.5-4: OSI-Referenzmodell und ONC-Netzwerkprotokolle in einem UNIX-Betriebssystem

Ethernet beschreibt die Leistungen des LAN (Local Area Network) auf Bitübertragungs- und Datensicherungsschicht. Die Hauptaufgabe der Vermittlungsschicht (Network Layer) ist die Verkehrslenkung des Netzwerks. Nachrichten sollen über das Internet-Protokoll (IP) auf optimalem Weg ihr Zielsystem erreichen. Der Adressierungsmechanismus dieser Ebene basiert auf der Internet-Adresse. Die Transportschicht übernimmt den Transport von Nachrichten zwischen den Kommunikationspartnern, steuert den Datenfluß und definiert die Transportqualität (gerichtet / ungerichtet) der Datenübertragung. Das TCP (Transmission Control Protocol) beinhaltet verbindungsorientierte Dienste. Jede Ankunft eines TCP-Protokolls muß beantwortet werden.. Die Kommunikationssteuerschicht (Session Layer) steuert den Austausch von Nachrichten auf der Transportverbindung. Durch die Darstellungsschicht (Presentation Layer) wird die sog. Transfermatrix (Codierung für die zu übertragenden Daten) festgelegt. Die Anwendungsschicht stellt die Schnittstelle zum Anwendungsprozeß bereit. Protokollbeispiele für diese Schicht sind: Mail, FTP, Telnet, NFS, ...

Der langwierige ISO/OSI-Normungsprozeß hat dazu geführt, daß sich in weiten Bereichen andere Kommunikationsprotokolle durchgesetzen konnten, z.B.:

- IPX/SPX[69]
  IPX läuft auf der OSI-Schicht 3 (Vermittlung) und regelt den Transport von Daten zwischen den einzelnen Netzknoten sowie den Austausch von Statusinformationen. SPX ist auf der Schicht 4

[69] Protokollsammlung von Novell: Internet Package Exchange / Sequential Package Exchange

(Transportschicht) angesiedelt und besorgt Auf- und Abbau von Verbindungen zwischen Client-PCs und Server-Anwendungen

- SNA[70]
- TCP/IP[71]

  Im Gegensatz zum OSI-Referenzmodell (und auch zu SNA) weist diese Kommunikationsschnittstelle nur 4 Schichten auf: Netzwerk, Internet, Transport und Anwendung. TCP/IP läßt sich auf verschiedenen Trägermedien (serielle Leitungen, X.25, Ethernet, usw.) einsetzen und ist vor allem im UNIX-Bereich weit verbreitet. Die Familie der TCP/IP-Protokolle hat sich bei Weitverkehrsnetzen (WAN) als Quasi-Standard durchgesetzt. Weltweit sind ungefähr zwei Millionen Rechner über das Internet[72] verbunden.

Die Unterstützung der verschiedenen Netzwerkmodelle (z.B. TCP/IP) und Betriebssysteme (z.B. Windows, Unix) erfolgt durch die einzelnen Software-Hersteller der DBS (Oracle, Sybase, Informix) bzw. durch die Werkzeug-Hersteller (z.B. Oracle, Microsoft), die Komponenten (z.B. ODBC-Treiber für besondere Plattformen und DBS) liefern.

Network-Operating-System

Das Network Operating System (NOS) hat die schwierige Aufgabe, den Service der im Netz erhältlich ist, als ein System darzustellen (single-system-image). Es setzt also die verteilten Teile des Netzwerkes zusammen und stellt sie als ein System dar - macht es also *transparent*.
NOS verwalten bspw. Datei- und Druckerressourcen und halten sie transparent, indem sie Anfragen über das LAN zu angeschlossenen Datei- und Druckerservern weiterleiten. Das NOS bietet dafür Proxies, die die Anfragen abfangen und an die entsprechenden Server weiterleiten.
Transparenz heißt also, dass das Netzwerk mit seinen Servern vor seinen Benutzern versteckt bzw. unsichtbar gemacht wird. Die wichtigsten Transparenz-typen sind:

- **Location transparency:** Es ist unwichtig, wo sich eine Ressource im Netz befindet, man muss z.B. nicht eine Ortsinformation in den Namen der Ressource mit eingeben.
- **Namespace transparency:** Man soll die gleichen Namensvereinbarungen (und namespaces) auf der lokalen oder irgendeiner Ressource im Netz verwenden können (unabhängig vom Ort, Typ oder Hersteller).
- **Logon transparency:** Es sollte ein einziges Passwort zum einloggen an allen Servern und Diensten die im Netzwerk verfügbar sind genügen.
- **Replication transparency:** Es ist unbekannt, wie viele Kopien einer Ressource existieren.
- **Distributed time transparency:** Keine Zeitunterschiede zwischen Servern.
- **Failure tranparency:** Das NOS soll uns von Netzwerk Fehlern abschirmen.

Kommunikation

Client/Server Applikationen sind über verschiedene Adressräume, physikalische Maschinen, Netzwerke und OS verteilt. Wie verläuft die Kommunikation?

[70] Vernetzungskonzept von IBM: System Network Architecture
[71] Transmission Control Protocol / Internet Protocol
[72] Weitverkehrsnetz, das etwa 4000 Netze mit über 6000 Rechnern verbindet

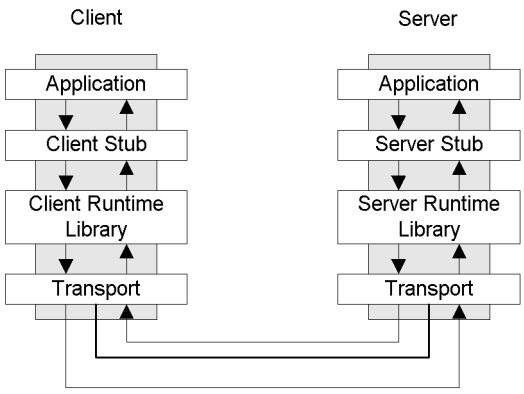
Alle NOS bieten ein „peer-to-peer“ Interface, welches eine Kommunikation der Applikationen untereinander ermöglicht und sehr nah an der physikalischen Verbindung angesiedelt ist.
Die meisten NOS bieten auch eine Form des Remote Procedure Call (RPC), der die physikalische Verbindung versteckt und den Server nur “einen Funktionsaufruf entfernt“ erscheinen lässt.
Eine Alternative zu RPC bietet Message Oriented Middleware (MOM), die Messages zur Abarbeitung in eine Warteschlange stellt.

1. Peer-to-peer Kommunikation
Das Protokoll ist symmetrisch, jede Seite kann die Kommunikation initiieren. Es maskiert nicht vollkommen die unterliegende physikalische Schicht. Das heißt, dass der Programmierer z.B. Übertragungstimeouts und Netzwerkfehler selbst abfangen muss.
Für eine einfache peer-to-peer Kommunikation wurde zuerst in Unix- Systemen und später auch in allen anderen OS *Sockets* eingeführt.
Sockets stellen logische “Steckdosen“ für die Herstellung von bidirektionalen Kommunikationsverbindungen bereit .
Der Socket-Kopf bildet die Schnittstelle zwischen dem Aufruf an das Betriebssystem zum Auf- und Abbau sowie Durchführung der Kommunikation und den weiter unten liegenden Systemschichten- dem Protokollteil und dem Gerätetreiber.
Die drei am meisten verwendeten Socket-Typen sind:

- Stream
- Datagram
- Raw

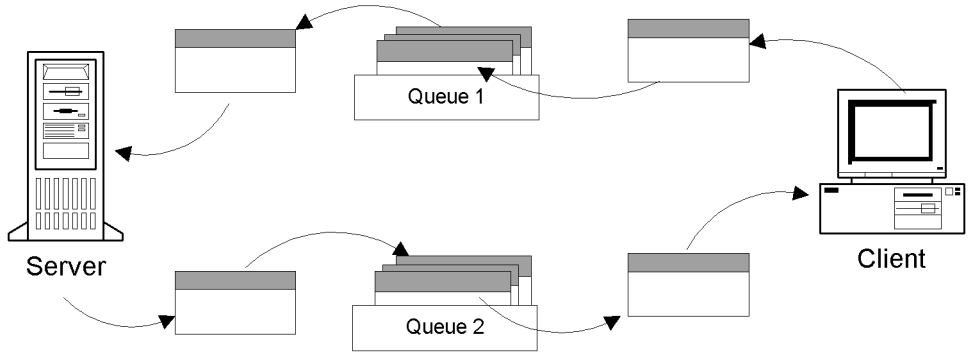
Stream Sockets setzen auf das TCP, **Datagram Sockets** auf das UDP und **Raw Sockets** auf das IP Protokoll auf (im Fall der TCP/IP Protokollfamilie). Der Typ der Sockets wird bei der Systemgenerierung festgelegt.
Eine Socketadresse für das TCP/IP Protokoll besteht aus der IP Adresse und der Portnummer. Die IP Adresse ist eine 32-Bit Zahl, die normalerweise durch vier Dezimalzahlen, getrennt durch einen Punkt, dargestellt wird. Der Port ist der Eingangspunkt zu einer Applikation und wird durch eine 16- Bit Integer Zahl dargestellt. Ports sind normalerweise Eingangspunkte zu Service der Server Applikationen. Wichtige kommerzielle Server Programme (z.B. Oracle DBMS, FTP) haben ihre festgelegten, sogenannten *well-known* Ports.

2. Remote Procedure Call (RPC)
Beim RPC ruft ein Client Prozess eine Funktion auf einem entfernten Server auf und verbleibt im Wartezustand bis er eine Antwort erhält - der RPC arbeitet also *synchron*. Parameter werden wie bei jeden gewöhnlichen Prozeduraufruf übergeben.
Es ist ein Client- und ein Serverteil des RPC nötig. Der Clientteil (Client-Stub) wird von der Client Applikation wie eine lokale Prozedur aufgerufen. Der auszuführende Prozedurcode befindet sich jedoch auf dem Server. Also werden die Parameter in eine Nachricht verpackt und an den Serverteil des RPC (den Server-Stub) gesendet. Dieser packt dann die Parameter wieder aus, führt mit diesen Parametern den Prozedurcode aus und sendet die Ergebnisse, wiederum in einer Nachricht verpackt, zurück zum Client-Stub. Die Rückgabedaten werden hier wieder ausgepackt und an die aufrufende Applikation übergeben.



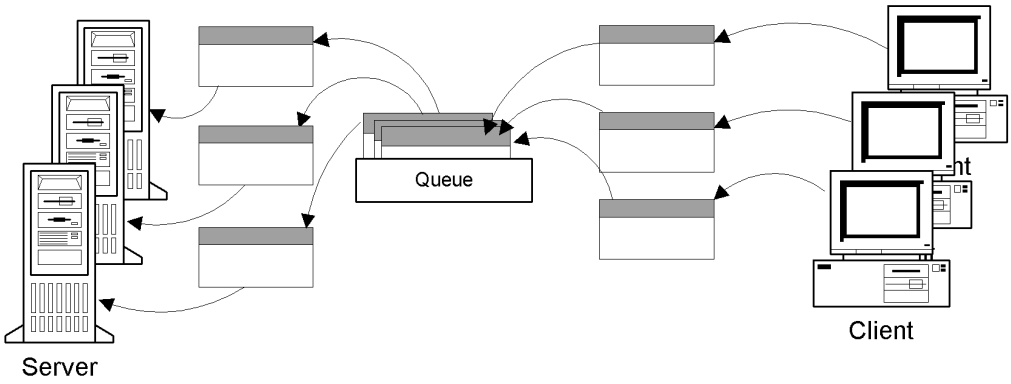
Die Schnittstelle des RPC muss in der Interface Description Language (IDL) deklariert werden, damit das Ein- und Auspacken korrekt funktioniert. Ein IDL-Compiler erzeugt daraus dann die Client- bzw. Server-Stubs. Die Implementierung der Prozedurrümpfe erfolgt wieder wie bei normalen Prozeduren.
Obwohl RPC das Leben eines Programmierers bereits erheblich vereinfachen gibt es einige Schwierigkeiten und Fehlerquellen zu beachten:
Wenn sehr viele Clients auf die Serverfunktionen zugreifen wird schnell ein großer Anteil der Rechenleistung des Servers nur für das Starten und Stoppen des Service, für die Priorisierung der Anfragen, Sicherheitsabfragen und Load-Balancing verbraucht.
Wie wird bei Fehlern reagiert? Ein recht umfangreicher Punkt, da beide Seiten separat ausfallen können. Es ist daher wichtig, dass die eingesetzte Software alle auftretenden Fehlerkombinationen abfangen kann. Wenn der Server nicht antwortet, blockiert der Client normalerweise. Nach einem Timeout muss er seine Anfrage wiederholen. Stürzt der Client nach einer Anfrage ab, muss der Server alle bis dahin vorgenommenen Veränderungen rückgängig machen können u.s.w.
Der Server muss garantieren, dass die gleichen Anfragen nur einmal abgearbeitet werden. Dies ist insbesondere bei mehreren Servern wichtig, wenn die Anfrage vielleicht an den zweiten Server abgesetzt wird, wenn der erste nicht reagiert.
Um mit RPC eine sichere und atomare Ausführung von Funktionen auch unter der Berücksichtigung aller möglichen Fehler zu gewährleisten ist ein erheblicher Entwurfs- und Implementierungsaufwand erforderlich.

3. Message Oriented Middleware (MOM)
Mit Hilfe von MOM Nachrichten und Warteschlangen können Clients und Server über ein Netzwerk kommunizieren, ohne dass sie über eine spezielle Verbindung fest miteinander verbunden sind- also *asynchron*. Clients und Server können so zu unterschiedlichen Zeiten laufen. Man kommuniziert einfach, indem man Nachrichten in eine Warteschlange stellt bzw. Nachrichten aus der Warteschlange entnimmt.



Dieses Prinzip löst alle Probleme, die beim RPC oder bei der peer-to-peer Verbindung aufgetreten sind:

- Der Client kann immer noch eine Anfrage senden, auch wenn der Server beschäftigt, runtergefahren oder gar nicht angeschlossen ist. Es muss nur die Warteschlange verfügbar sein, in die der Client seine Anfrage ablegen kann. Ist der Server wieder verfügbar, kann dieser dann die Nachrichten aus der Schlange abholen und bearbeiten.
- Umgekehrt kann auch der Server eine Antwort senden, wenn der Client runtergefahren oder nicht angeschlossen ist. Es muss wiederum nur seine Antwort- Warteschlange verfügbar sein. Der Server setzt seine Antwort einfach in diese Schlange. Ist der Client wieder verfügbar, prüft er in seine Warteschlange und findet die erwartete Antwortnachricht des Servers.
- Mehrere Server können ihre Anfragen der gleichen Warteschlange entnehmen und so die Arbeitslast auf diese verteilen. Sobald ein Server eine Anfrage abgearbeitet hat kann er die nächste aus der Schlange entnehmen und diese abarbeiten. Es kann also nicht passieren, dass ein Server überlastet ist, während sich der andere im Leerlauf befindet.
- Anfragen in der Warteschlange können für eine prioritätsgesteuerte Bearbeitung mit Prioritäten versehen werden.



Gängige MOM Produkte bieten ein einfaches API für viele verschiedene OS Plattformen. Außerdem bieten sie *persistente* und *nicht-persistente* Nachrichten-Warteschlangen. Nicht-persistente Warteschlangen werden im Arbeitsspeicher gehalten. Persistente Schlangen werden auf den Plattenspeicher geschrieben und sind dadurch langsamer, können jedoch nach einem Systemausfall wieder hergestellt werden. Die Warteschlangen können sich lokal auf dem Rechner befinden, oder entfernt über ein Netzwerk. Der Systemadministrator kann die Größe der Warteschlange einstellen.

1.5.3.2 Client-Server-Architekturen

1.5.3.2.1 Architekturformen

In der Praxis haben zahlreiche Realisierungen von Client-Server-Architekturen ergeben. Sie entsprechen weitgehend den bekannten Verteilungsformen[73] von Anwendungssystemen.

1. Disk-Server

Anforderungen des Dateisystems werden an den Disk-Server weitergeleitet und dort auf die Plattenperepherie des Servers umgesetzt

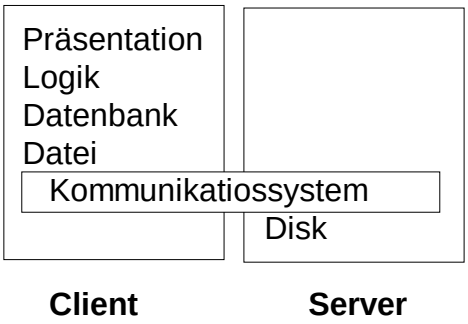


Abb. 1.5-5: Disk-Server

Es gibt nur einen physikalischen Plattenspeicher, der von allen Clients benutzt werden kann. Eine spezielle Anwendung des Disk-Servers ist der "Remote-Boot" für Arbeitsplatzrechner ohne Diskettenlaufwerk (diskless client). Das Betriebssystem wird dabei vom Disk-Server geladen.
Bei einem Disk-Server können zwar zahlreiche Leser, aber nur ein einziger Schreiber auf die gemeinsam genutzte Ressource zugreifen. Das ist für das Laden von Programmen oder das Lesen von gemeinsamen Daten ausreichend.

2. File-Server

Der File-Server empfängt alle Dateianforderungen, die von einzelnen Clients (z.B. PCs und Workstations) an ihn gesandt wurden.

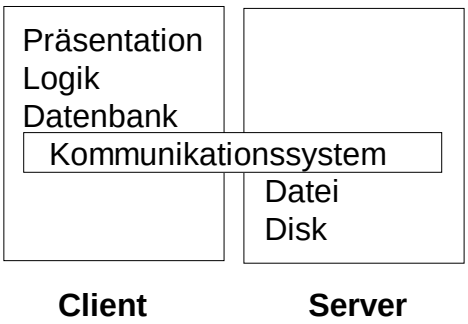


Abb. 1.5-6: File-Server

[73] Hans Joachim Petzold, Hans Jochen Schmitt: Verteilte Anwendungen auf der Basis von Client-Server-Architekturen in HMD 170/1973, S. 79 - 91

Operationen, z.B. das Sperren von Bereichen werden zentral im File-Server verwaltet. Mehrere Anwendungen können konkurrierend auf gemeinsame Datenbestände zugreifen. Es gibt u.U zahlreiche unabhängige Datenbanksysteme jedoch nur ein Dateisystem. Die einzelnen Workstations sind mit dezentraler Intelligenz ausgestattet, so daß alle Programme im lokalen Arbeitsspeicher ablaufen. Zur Inanspruchnahme bestimmter Dienste werden Programmaufrufe über die in einer Workstation eingebaute Netzwerkkarte an den Server weitergeleitet.
Nach dem Prinzip des File-Server arbeiten die meisten Netzwerkbetriebssysteme. Bekanntestes Produkt ist zur Zeit Novell Netware. Mit Hilfe sog. Netware Loadable Moduls (NLM) läßt sich das Netzwerk an verschiedene Architekturen[74] anpassen.
Den restlichen Markt für Netzwerbetriebssysteme teilen sich LAN-Manager bzw. LAN-Server[75] und Banyan Vines.

[74] Viele Systemdienste wie etwa die ganze Schichtung möglicher Netzwerkprotokolle (IPX, ZCP/IP usw.) wurden auf der Server-Seite als NLM realisiert

[75] Der LAN-Manager kommt von Microsoft, der LAN-Server von IBM. Der Ansatz ist aber gleich. Der Server-Teil setzt auf OS/2 als Betriebssystem. Nachfolger des LAN-Manager ist inzwischen der Windows NT Advanced Server, der Netzwerkfunktionen von Windows NT benutzt

Netware File Server

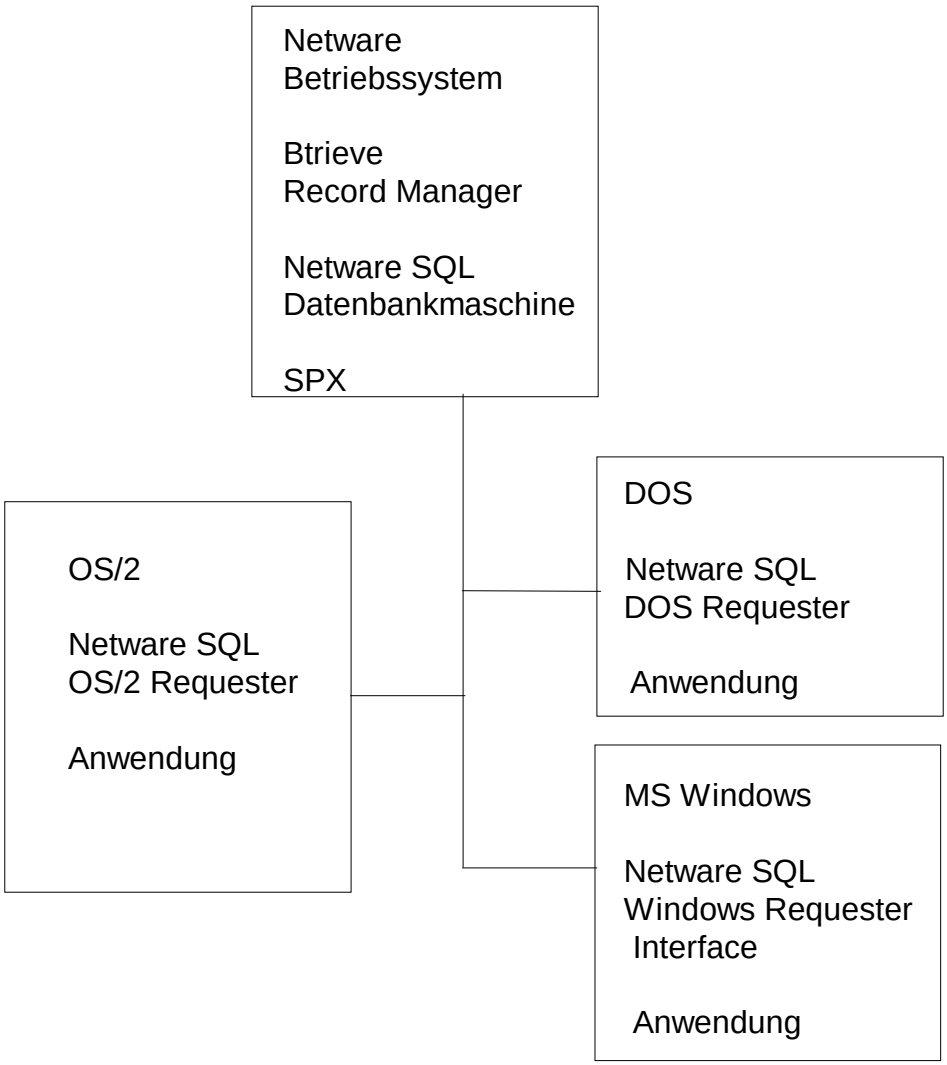


Abb. 1.5-7: Client-Server Architektur des Netware SQL

Btrieve ist die Zugriffsmethode von Netware SQL. Geeignete „client requester"-Programme kommunizieren über SPX (Sequenced Paket Exchange Protocoll) mit Netware SQL.
Auch UNIX-Server sind (auch in PC-Netzen) weit verbreitet. Für alle UNIX-Abkömmlinge gibt es NFS (Network File System)[76]. Es basiert auf dem TCP/IP-Netzwerkprotokoll. Heterogene Netzwerke bieten in diesem Zusammenhang keine grundsätzlichen Probleme. Wegen der in UNIX-Betriebssystemen enthaltenen unbegrenzten TCP/IP- und NFS-Lizens kann das Netzwerk beliebig wachsen.

3. Datenbank-Server

Datenanforderungen werden bereits an der Schnittstelle zum DB-System (z.B. in Form von SQL-Anweisungen) abgefangen. Das Datenbanksystem mit allen untergeordneten Schichten befindet sich auf dem Server.

[76] eine Entwicklung von Sun Microsystems, es gibt auch ein PC-NFS von Sun

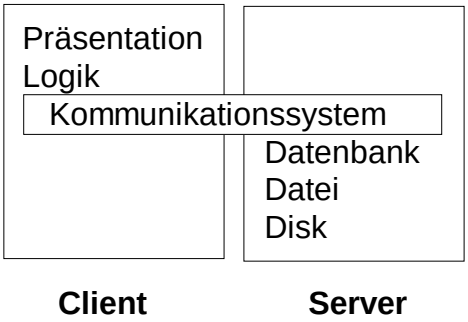
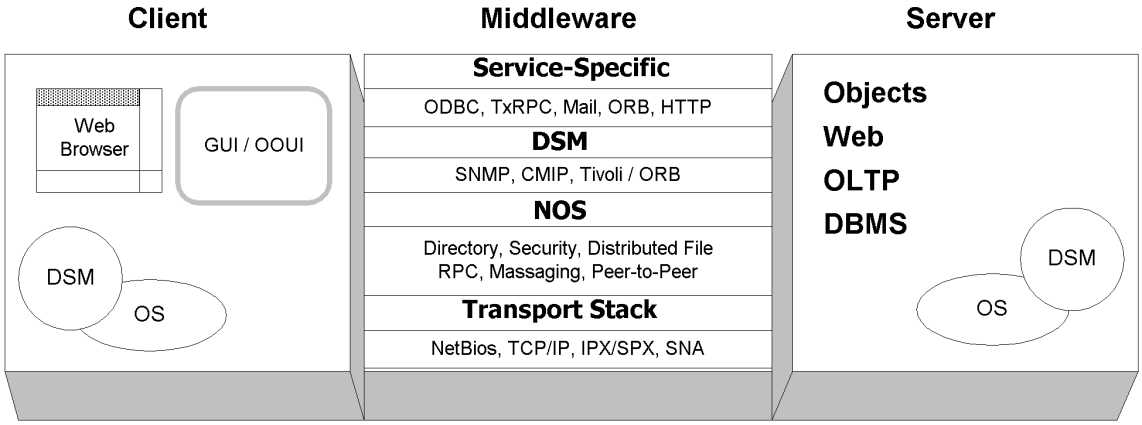


Abb.1.5-8 Datenbank-Server

Bei der Datenverarbeitung im Netzwerk werden Programme (, die lokal im Netzwerk zur Verfügung stehen,) in den Hauptspeicher des Client geladen und ausgeführt. Erhalten diese Programme Datenbankzugriffe, dann sorgt eine spezielle Softwareschicht, die „Middleware“, dafür, daß alle Zugriffe auf einen Datenbankserver „umgelenkt“ werden. Im Rahmen der 3-Tier-Architektur wird das Client/Server-Modell in drei große Blöcke unterteilt: Client, Middleware und Server.
Im Client Block läuft der Client Teil der Applikation auf einem Betriebssystem, welches ein Graphical User Interface (GUI) oder ein Object Orientet User Interface (OOUI) bietet und Zugriff zum verteilten Service hat. Das Betriebssystem arbeitet mit der Middleware zusammen und überlässt ihr die Bearbeitung der nicht lokalen Servicees. Außerdem läuft auf dem Client eine Komponente des Distributed System Management (DSM).
Im Server Block läuft die Server Seite der Applikation. Die Serversoftware läuft meist auf einem kompletten Server Software Package. Die wichtigsten Serverplattformen sind dabei: SQL Datenbank Server, TP Monitore, Groupware Server, Objekt Server und Web Server. Das Betriebssystem stellt das Interface zur Middleware bereit und liefert die Anfragen für Service. Auch auf dem Server läuft eine Komponente des DSM.
Der Middleware Block läuft auf der Client- und der Serverseite der Applikation und kann wieder in drei große Kategorien unterteilt werden: Transport Stacks, Netzwerk Operating System (NOS) und Service-spezifische Middleware. Auch der Middleware Block hat eine DSM Komponente.



Middleware ist ein recht ungenauer Begriff, der all die verteilte Software beinhaltet, die benötigt wird um Interaktionen zwischen Client und Server zu

ermöglichen; oder anders gesagt: Ist die Verbindung, die dem Client ermöglicht einen Service vom Server zu erhalten.
Middleware beginnt beim Client API, über welchen ein Serviceaufruf abgesetzt wird, beinhaltet die Übertragung des Aufrufes über ein Netzwerk und die resultierende Antwort auf den Aufruf. Sie beinhaltet nicht die Software die den eigentlichen Service bietet (das befindet sich im Server Block) oder das User Interface (welches sich im Client Block befindet).
Der Datenbankserver berechnet das Ergebnis und das Programm auf dem Client wird automatisch mit diesen Daten versorgt. Im Unterschied zum Dateiserver führt der Datenbankserver komplexe Operationen aus.
Eine Transaktionsverwaltung ist über den Datenbank-Server möglich. Es können auch Server eingesetzt werden, die eine andere Betriebssystem- oder Hardwarearchitektur haben als Client-Rechner. So sind spezielle Datenbankrechner als Datenbank-Server einsetzbar, sobald sie über ein Kommunikationssystem von den Clients aus erreicht werden können. Bekannte Produkte sind: Oracle, Ingres, IBM Database Manager sowie Server von SyBase und Gupta Technologies.

Das Oracle Client-/Server-Konzept
Ab Version 7 heißt das relationale Datenbanksystem der Firma Oracle „Oracle7Server“. Dieser Begriff betont ab Version 7 ausgeprägte Serverfähigkeiten der Datenbankssoftware von Oracle.
Auch dieser Datenbankserver muß in irgendeiner Form (mit SQL-Befehlen) versorgt werden. Dies übernimmt ein Client-Prozeß, der auf dem gleichen Rechner läuft wie der Oracle7Server (lokale Anwendung) oder auf einem anderen Rechner im Netzwerk (Client-/Server-Anwendung).
Eine wichtige Aufgabe des Client ist die Identifikation der Anwendung, die ausschließlich über das User Programmatic Interface (UPI) an den Datenbankkern auf den Server gesandt wird.

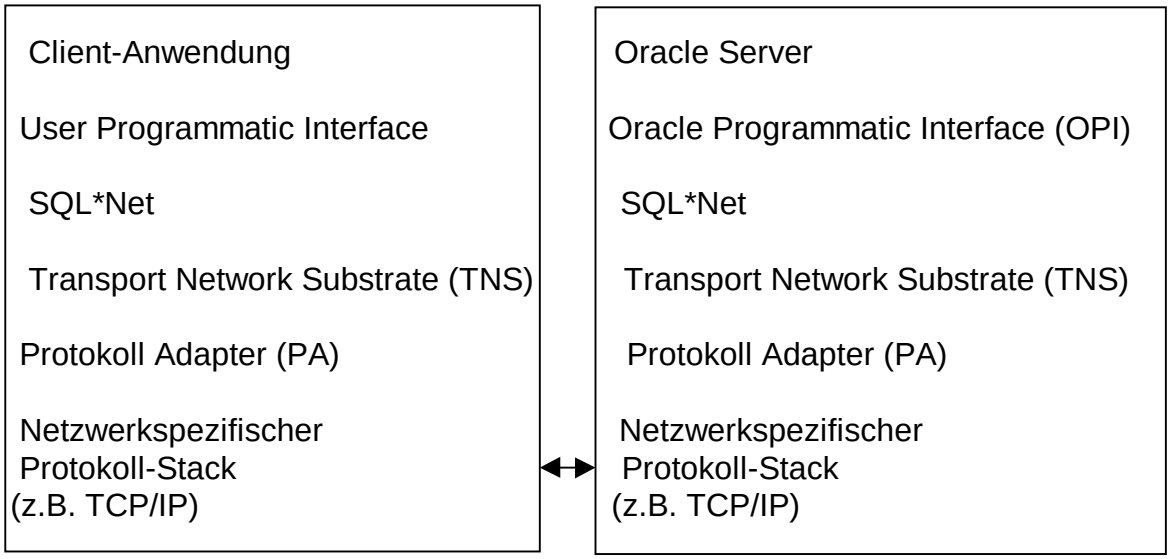


Abb. 1.5-9: Die Komponenten bei einer Client-/Server-Verbindung

UPI ist eine Schicht der Client-Anwendung mit einem Satz von Funktionen, über die ein SQL-Dialog zwischen Client und Server durchgeführt werden kann. Die Kontrolle wird dann an SQL*Net weitergegeben, das für den Transport der Daten zuständig ist. In Client-/Server-Verbindungen sorgt **SQL*Net** für die Verbindung zwischen Client und Server.
SQL*Net ist die Middleware von Oracle. Auf der Client-Seite besteht SQL*Net im wesentlichen aus einigen Bibliotheken und Konfigurationsdateien. Auf der Serverseite gibt es zusätzlich einen Listener-Prozeß, der auf Verbindungen wartet und entsprechende Verbindingen aufbaut. Das Interface der Anwendungen und des Oracle7Server zu SQL*Nert ist protokollunabhängig.
SQL*Net ist schichtenweise strukturiert. Die unterste Schicht bilden netzwerkspezifische Protokolle, z. B. TCP/IP, IPX/SPX. Darauf folgt die „Protocol Adapter (PA) – Schicht“. Auf diese Schicht baut die zentrale Komponente, Transport Network Substrate (TNS) auf. TNS realisiert an seiner Schnittstelle elementare Kommunikationsfunktionen und gibt seine Informationen an die Protokoll Adapter weiter.
TNS soll eine einheitliche Schnittstelle erzeugen. Applikationen, die auf TNS aufsetzen, können völlig unabhängig von spezifischen Protokollen implementiert werden. Auf TNS setzen zur Zeit zwei Produkte auf: SQL*NET und das Multiprotocoll Interchange. SQL*NET ist die Schnittstelle auf die Clients bzw. Server aufsetzen. Das Multiprotocol Interchange dient zur Protokollumwandlung und ermöglicht, daß Client und Server mit unterschiedlichen Protokollen betrieben werden können.
Im Bereich des Kommunikationsteils auf dem Client gibt die Software-Komponente SQL*Net[77] alle notwendigen Informationen an TNS, das Daten an den Prokoll-Adapter (PA) weiterleitet, der für den protokollspezifischen Transport zuständig ist. Das Netzwerk-Protokoll transferiert die Daten über das physikalische Netz zum Server. Dort angekommen nehmen die Daten den

[77] Der Begriff SQL*Net wird für zwei Dinge verwendet: Einerseits als Oberbegriff für alle Netzwerkprotokolle und die Middleware von Oracle, andererseits als Bezeichnung für diejenige Softwarekomponente, die auf Client- und Serverseite installiert ist.

umgekehrten Weg über PA, TNS und SQL*Net. Das Multiprotocoll Interchange des Server verfügt über eine Listener (TNS-), der ankommende Verbindungen auf das Ziel überprüft und ann einen Serverprozeß zur Abarbeitung der SQL-Anweisungen startet. Von der SQL*Net-Schicht werden Daten an das Oracle Programmatic Interface (OPI) weitergegeben (arbeitet entgegengesetzt zum UPI). Für jeden Aufruf des UPI gibt es im OPI eine Funktion, die die angeforderte Aufgabe erfüllt. Oberhalb des OPI setzt der Server auf, der die Anforderungen des Client erfüllt und die Ergebnisse über das OPI an den Client zurücksendet.

SQL-Server
SQL-Server bauen auf der "Client-Server-Architektur" auf. Der Datenbankserver (database server) läuft auf einem eigens dafür vorgesehenen Rechner und bietet die notwendigen Datenbankfunktionen für alle Netzteilnehmer an. Der Datenbankserver ist auf die Sprache **SQL** ausgerichtet. Der Client fordert mit einer SQL-Anfrage eine Ergebnistabelle an, die er dann weiterverarbeitet. Der Server ist in der Lage, SQL-Anfragen auszuwerten und ganze Tabellen zu übergeben.
SQL-Server haben einen "transaction manager" zur Synchronisation von Tabellen und Indexverzeichnissen (z.B. bei einem Absturz). Dauerhaft enthalten Tabelle und Indexverzeichnisse die Resultate von ausschließlich abgeschlossenen Transaktionen. "Backup/Restore-Utilities" generieren Kopien und sorgen für die Wiederherstellung der Datenbank (z.B. bei Plattenausfall). Die vollständige Wiederherstellung aller Aktualisierungen zwischen der letzten Sicherung (Backup) und dem Zeitpunkt des Plattenfehlers übernehmen "Recovery-Prozeduren". SQL-Server unterstützen Entitäts- und Beziehungsintegrität [78] , automatische Seiten- oder Datensatzsperre und verfügen uber eine automatische Erkennung von Verklemmungen.
Der Server erledigt die Arbeit, der Front-End zeigt Benutzerfreundlichkeit. Front-Ends von Tabellenkalkulationen oder "Multimedia-Programme" sollen auf verschiedene DBMS zurückgreifen können. Ein genormtes SQL stellt (zumindestens theoretisch) sicher, daß die Kommunikation zwischen beiden reibungslos funktioniert. In der Praxis verlangt jedoch jedes DBS eine Sonderbehandlung [79].

4. Präsentations-Server

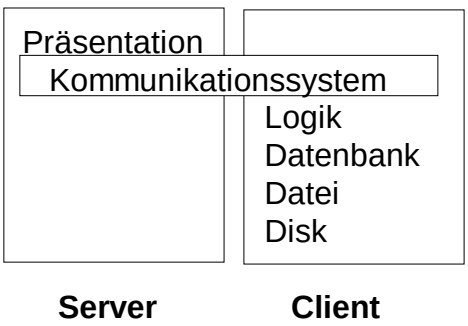


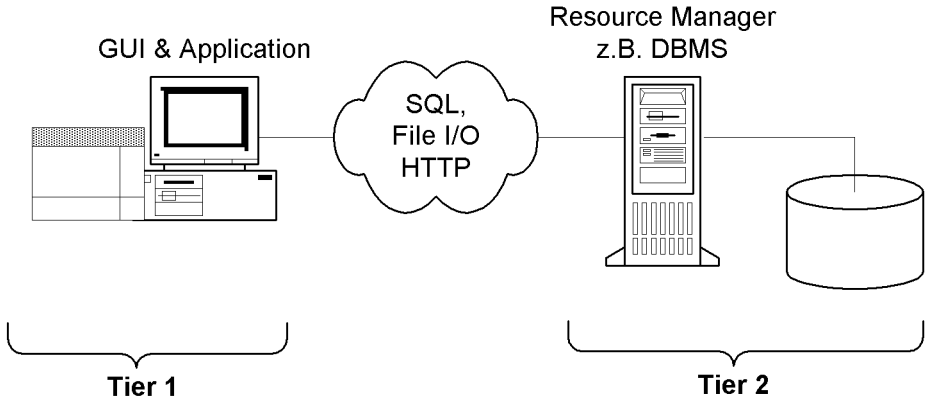
Abb. 1.5-10: Präsentationsserver

[78] vgl. 1.2.6
[79] vgl. ODBC (Open Database Connectivity)-Interface von Microsoft, das im wesentlichen eine Windows-API ist. Für jedes DBMS gibt es einen ODBC-Treiber.

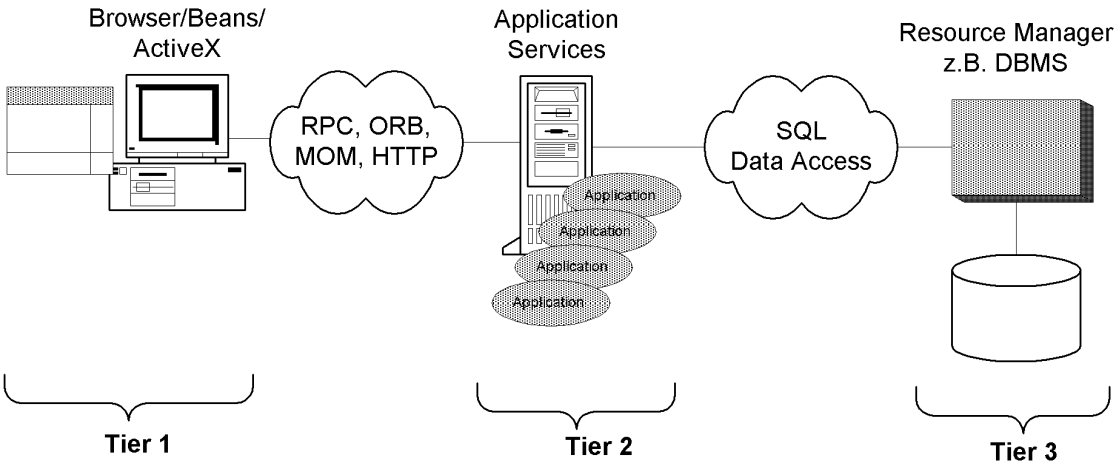
Diese Form der Anwendung wird auf Unix-Systemen mit dem X-Window-Ansatz[80] verfolgt. Der den Dienst der Präsentation anbietende Arbeitsplatzrechner ist hier der Server. Der Client bedient sich dieser Dienste zur Interaktion mit dem Benutzer.

1.5.3.2.2 2-Tier und 3-Tier Client-/Server-Architekturen

Bei **2-Tier Client/Server Systemen** befindet sich die Applikationslogik entweder im User Interface des Clients oder in der Datenbank des Servers.
Beispiele für 2-Tier Client/Server Architekturen sind: File Server und Datenbank Server mit gespeicherten Prozessen.



Bei **3-Tier Systemen** befindet sich die Applikationslogik in der Mittelschicht, getrennt von den Daten oder dem User Interface. Sie kann somit getrennt vom GUI und der Datenbank gewartet und entwickelt werden.
Beispiele für 3-Tier Client/Server Architekturen sind: TP-Monitore, Object Transaction Monitore, Web Application Server.



[80] X-Window-Ansätze realisieren die Window-Manager OSF-Motif, Open Look

Vergleich von 2- und 3-Tier-Architektur: Der größte Vorteil der 2- Tier Architektur ist, dass Client/Server Applikationen sehr einfach und schnell erstellt werden können, z.B. mittels visuellen Programmiertools. Diese Applikationen arbeiten meist noch recht gut in Prototypen und kleineren Installationen, zeigen aber schnell ihre Grenzen wenn sie in größeren Anwendungssystemen mit einer hohen Benutzerzahl und vielen konkurrierenden Zugriffen eingesetzt werden.
3- Tier Architekturen sind besser skalierbar, robust und flexibel. Sie können Daten von mehreren verschiedenen Quellen verarbeiten, sind im Netzwerk einfacher zu administrieren und zu entwickeln da der meiste Code auf dem Server läuft. 3- Tier Applikationen verringern die Netzwerklast durch zusammengefasste Serviceaufrufe. Anstatt mit der Datenbank direkt zu kommunizieren wird mit der Mittelschichtlogik auf dem Server kommuniziert. Diese arbeitet dann eine Reihe von Datenbankzugriffen ab und reicht nur das gewünschte Ergebnis zurück. Daraus resultiert eine höhere Performance durch wenige Serveraufrufe.
Außerdem wird eine höhere Sicherheit gewährleistet, da kein direkter Zugriff auf die Datenbank durch den Klienten erfolgt.

1.5.3.3 Client/Server und Internet / Intranet

Grundlage der Client-/Server-Architektur ist eine Verteilung zwischen verschiedenen Systemen und Systemteilen. Die Internet-Technologie[81] ist eine Hardware- und Systemstruktur, kombiniert mit einheitlichen Protokollen und Präsentationsformaten. Sie repräsentiert eine geschickte Art der Praktizierung von Client-/Server-Architekturen.

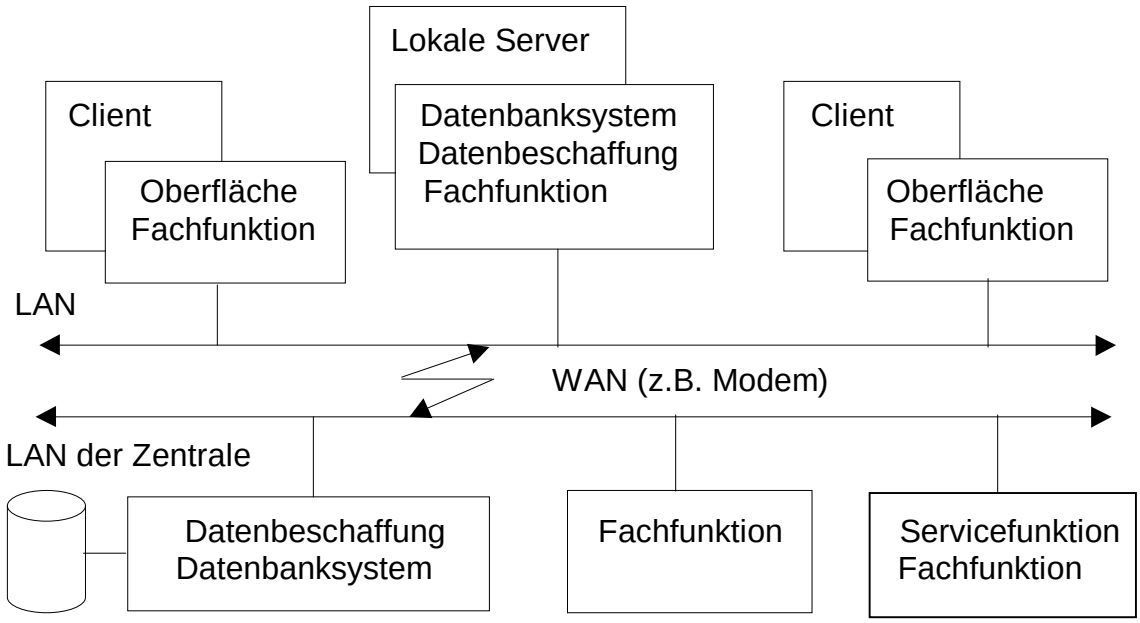


Abb. 1.5-11: Aufgabenverteilung auf verschiedenen Schichten (Präsentation, Funktionen, Datenbeschaffung /
Datenbank, Services) des Client-/Server-Modells

Die Abb. zeigt die Verteilung der Aufgaben auf verschiedene Schichten:
Die Präsentationsschicht übernimmt die Darstellung der Information gegenüber dem Benutzer (klassische Frontend-Bearbeitung). Die Funktionsschicht umfaßt die fachlichen Funktionen, unabhängig von Darstellung und Speicherung. Die Datenbeschaffungsschicht umfaßt die Speicherung der Daten auf den erforderlichen Medien mit den entsprechenden Systemen. Die Service-Schicht stellt Werte und Funktionen in allgemeiner Form bereit, die systembezogen (oder umgebungsbezogen)[82] sind.
Zur Nutzung des Internet steht eine Vielzahl von Anwendungen zur Verfügung, die als Internet-**Dienste** bezeichnet werden. Die Dienste[83] stützen sich auf ein Client-/Server-Modell ab. Bekannte Dienste im Internet sind: Telnet, File Transfer Protocol (FTP), Usenet News, World Wide Web (WWW).
Alle Internet-Dienste stützen sich auf das Client-/Server-Konzept. Das Client-Programm (Client) stellt die Schnittstelle zwischen Benutzer und Server-Programm (Server) dar. Der Server bietet Informations- und

[81] firmenintern als Intranet genutzt
[82] Das fängt beim Tagesdatum an und reicht zu so komplexen Funktioen wie bspw. Zeitsynchronisierung innerhalb eines komplexen Netzes oder der Nachrichtenvermittlung zwischen Objekten (Object Request Broker.
[83] Für eine Vielzahl von Diensten sind Standqardisierungen in sog. RFCs (Request for Comments) niedergelegt

Kommunikationsvermittlung an. Aufgabe des Client ist es, Anfragen des Benutzers in eine maschinenverständliche Art „umzuformulieren“ und dem Benutzer die vom Server gelieferte Antwort zu präsentieren. Für die Benutzung eines Internet-Dienstes ist ein Client- und ein Server-Programm nötig. Der Client übernimmt Vorfeldaufgaben (front-end application) der Informationsverarbeitung und erlaubt unterschiedliche Datenquellen unter einer Oberfläche zu integrieren.
Internet-Dienste ermöglichen die Kommunikation mit anderen Internet-Teilnehmern, die Nutzung von Informationsressourcen im Internet und das Anbieten von Informationen über das Internet. Der Internet-Dienst, dem das Internet das exponentielle Wachstum verdankt, ist das World Wide Web (`WWW`).
Die Idee des **World Wide Web** (WWW oder einfach nur Web) ist Anfang 1989 am CERN[84] entstanden. Auslöser waren Schwierigkeiten beim Auffinden relevanter Informationen im Internet in Hard- und Software. Zur Lösung des Problems wurde ein auf der Client-/Server-Architektur aufbauendes, hypertext-basiertes System vorgeschlagen. Hypertext wird dabei als Weg verstanden, Informationen derart miteinander zu verknüpfen, daß sich der Benutzer nach eigenem Belieben durch die Informationen bewegen kann. Die Verknüpfungen erscheinen dabei als Knoten in einem verwobenen Netz.

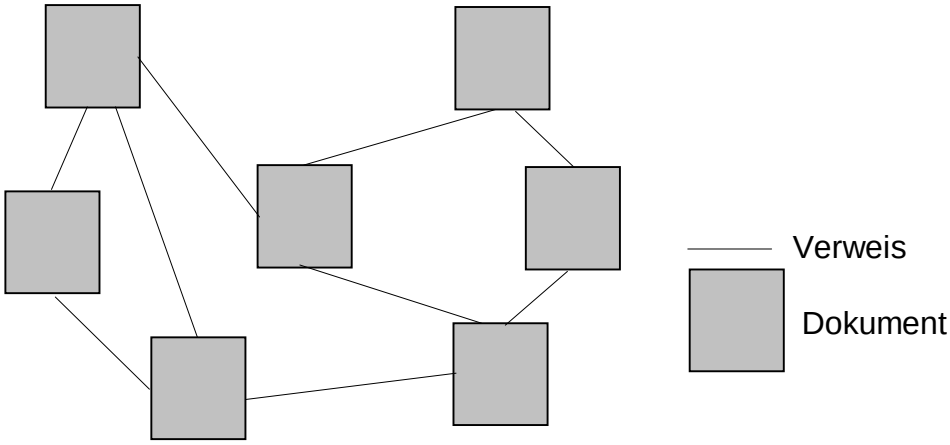


Abb. 1.5-12: Struktur des World Wide Web

Eine verteilte Hypermedia-Anwendung kann als Netz angesehen werden, dessen Knoten Text, Grafik, Ton darstellen. Verbindungen (Links) kennzeichnen die Beziehungen zwischen den Knoten und werden von der Systemsoftware verwaltet. Die möglichen Verbindungen werden im Dokument (Fenster) an der entsprechenden Stelle gekennzeichnet. Der Benutzer kann durch Anklicken derartiger Markierungen im Dokument oder durch Weiterblättern bzw. Rücksprung zu einem früheren Knoten fortfahren, solange er will.
Dokumente werden in einem speziellen Format, dem Hypertext-Format gespeichert. Zum Erstellen von Dokumenten steht die Seitenbeschreibungssprache `HTML` (`Hypertext Markup Language`) zur Verfügung[85].

[84] europäisches Zentrum für Teilchenphysik bei Genf
[85] bezieht sich auf die in ISO 1986 standardisierte SGML (Standard Generalized Markup Language).

Die wichtigsten HTML-Steueranweisungen sind Hyperlinks. Damit kann man auf Dokumente verweisen, die irgendein anderer Rechner im Internet bereitstellt. Diese Verweise bilden ein weltumspannendes Netz, das WWW.

Das **Hypertext Transfer Protocol** (HTTP) dient zur Übertragung von Informationen aus dem World Wide Web (WWW). HTTP ist ein objektorientiertes Protokoll zur einfachen Übertragung von Hypertext-Dokumenten zwischen Client und Server.

Client-Programme, die HTTP benutzen, werden (in der Regel) als **Web-Browser**, Server-Programme **als Web-Server** bezeichnet. Der Browser schickt an den Server die Aufforderung eine bestimmte HTML-Seite zu übertragen. Falls er in dieser Seite weitere Verweise (z.B. auf Bilder) entdeckt, schickt er weitere Übertragungswünsche hinterher. Das Besorgen der gewünschten Dokumente erfolgt über ein einheitliches Adreßschema, dem Uniform Resource Locator (URL), durch den Internet-Standort und die Art der zu übertragenden Information identifiziert werden. Der URL besteht im wesentlichen aus zwei Teilen:

- Er wird mit der Bezeichnung des Übettragungsprotokolls eingeleitet, z.B.: „http“.
- Danach folgt (mit zwei vorangestellten //) der Name des Internet-Rechners, auf dem die gewünschte Information gespeichert ist. Wird eine bestimmte Datei in einem bestimmten Verzeichnis referenziert, dann werden noch Verzeichnisname und Dateiname angefügt.

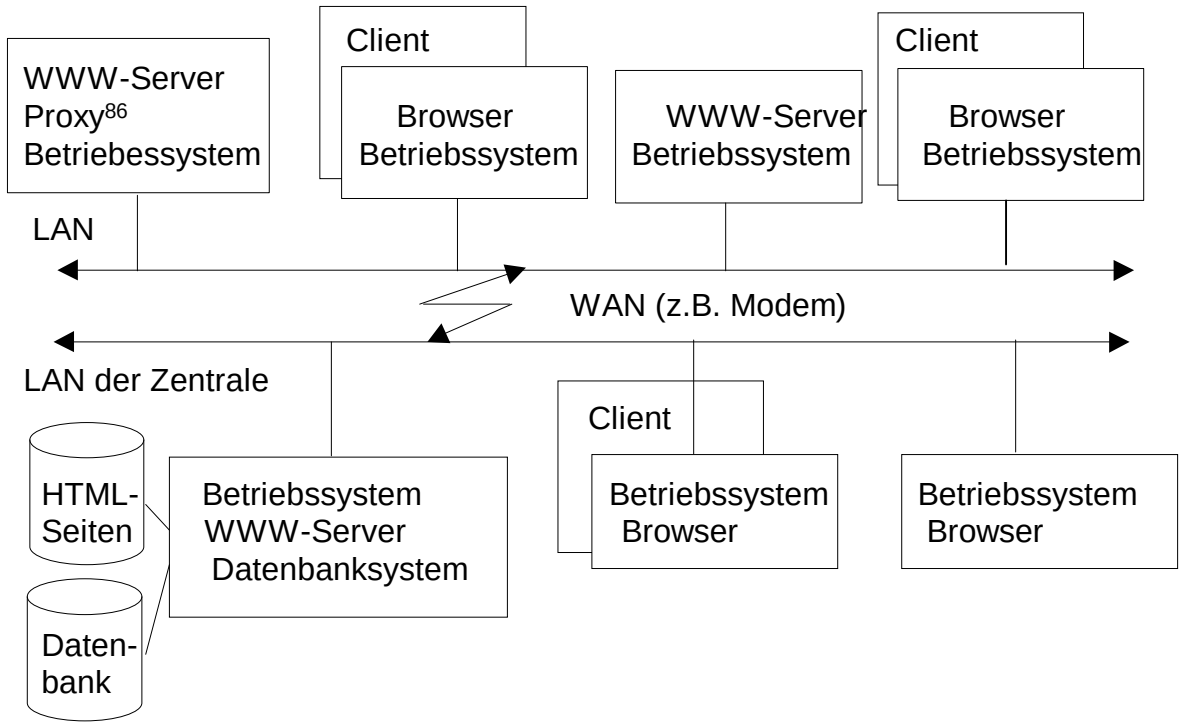


Abb. 1.5-13: Intranet-Architektur

Bei der Client-/Server-Kommuniktaion über das Internet benötigt man eine Server, der ständig in Bereitschaft ist und auf Anfragen von einem Client wartet. Der Client baut eine Verbindung auf, indem er eine Anfrage an den Server richtet und dieser die Verbindung aufnimmt. Das einfachste Bsp. hierfür ist ein Web-Server, der auf einem Host-Rechner installiert ist. Der Web-Browser fungiert dabei als Client, der eine Verbindung zu einem bestimmten Web-Server für den Datenaustausch herstellt. In erster Linie wird der Client HTML-Dokumente vom Web-Server empfangen.
Viele Web-Knoten legen jede Web-Seite als eigene HTML-Datei ab. Solche Seiten sind allerdings statisch, d.h. sie erzeugen bei jedem Aufruf denselben Inhalt. In vielen Unternehmen steigt aber das Interesse, vorhandene, in Datenbanksytemen gehaltene Dokumente und Informationen für Web-Clients verfügbar zu machen. Die Anbindung einer Datenbank an einen Web-Server erleichtert die Aktualisierung der Inhalte. Allerdings müssen dazu die Daten mediengerecht aufbereitet, d.h. in HTML, der Seitenbeschreibungssprache des Web verpackt werden. Das geschieht mit Hilfe von Programmen, die Daten aus der Datenbank auslesen und aus den so gewonnenen Informationen HTML-Dokumente erzeugen. Die zugehörigen Programme laufen auf dem Web-Server, von dem auch die statischen Dokumente geladen werden.

[86] Proxy-Server: Das ist ein zusätzliches lokales Datenbeschaffungsinstrument für einen bestimmten anwenderkreis (Clients des oberen LAN). Die Clients dieses Anwenderkreises gehen nicht direkt ans Netz, sondern beauftragen ihren Proxy über die schnelle lokale Verbindung, eine oder mehrere Informationsseiten zu beschaffen. Der Proxy holt sich die Seite, speichert sie in einem lokalen Cache-Speicher und liefert sie an den lokal anfordernden Client aus.

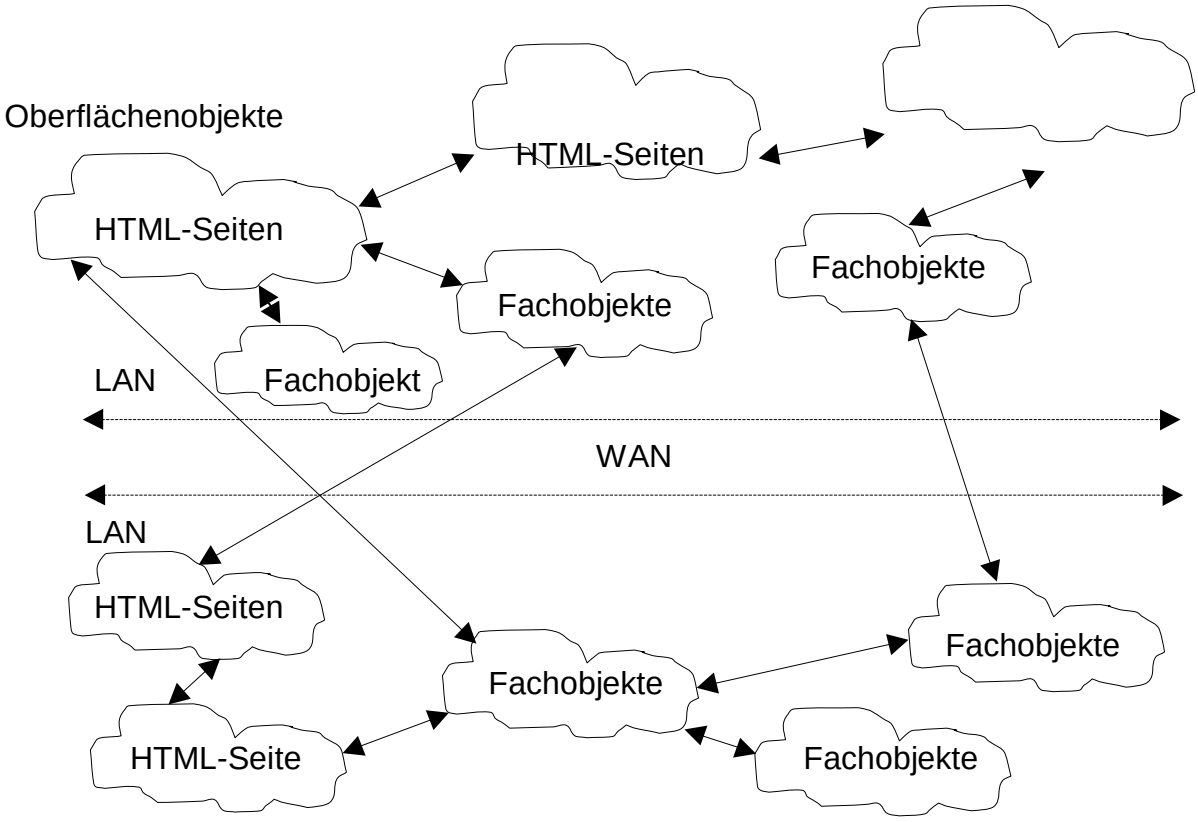


Abb. 1.5-14: Aufbau und Kommunikation der Objekte im Internet

Typischerweise werden Applikationen in 3 Aufgabenbereiche unterteilt: Datenhaltung, Anwendung(slogik) und Präsentation[87]. Diese 3 Aufgabenbereiche müssen nicht auf einem Rechner oder von einem Prozeß wahrgenommen werden, sondern können auf mehrere Rechner bzw. Prozesse verteilt sein.

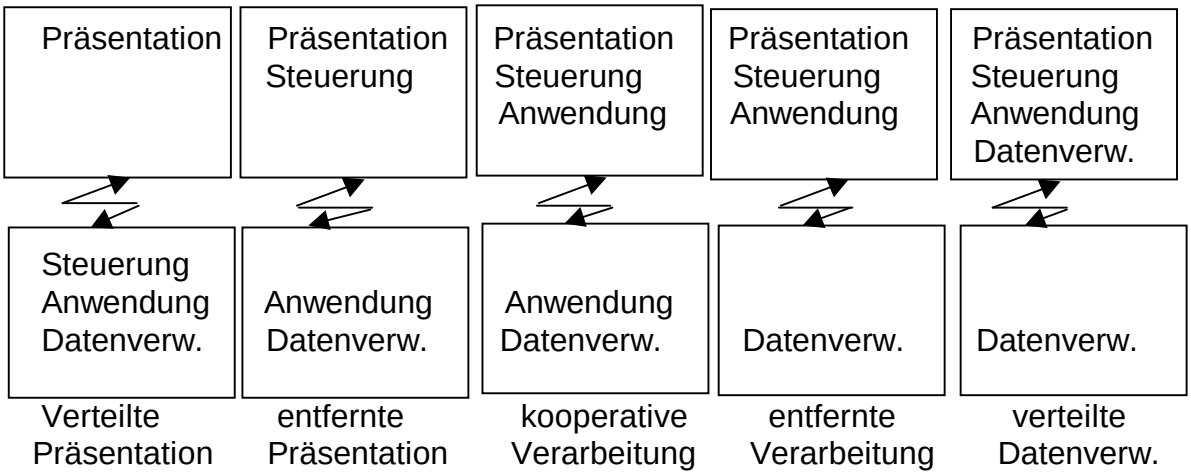


Abb. 1.5-15: Verteilungsmöglichkeiten im Client-/Server-Modell

Bei der entfernten Repräsentation (z.B. Common Gateway Interface (`CGI`)) findet die Verarbeitung „Server“-seitig statt, auf dem Client werden nur Repräsentationsaufgaben ausgeführt.

[87] Vgl. Client-Server-Architekturen

Bei der verteilten Datenverwaltung werden die Aufgaben zwischen Client und Server aufgeteilt. Beispiele sind Entwicklungsumgebungen, die sowohl auf lokale als auch entfernte Datenbanken zugreifen können. Die verteilte Datenverwaltung geht tief in Theorie und Konzept verteilter Datenbanken ein.
Bei der kooperativen Verarbeitung findet die Datenhaltung Server-seitig statt. Repräsentationsaufgaben werden vom Client wahrgenommen und die Applikations-Funktionsschicht wird zwischen Server und Client aufgeteilt.
Bei der entfernten Verarbeitung wird die Datenhaltung vom Server wahrgenommen. Die Verarbeitung findet „Client"-lastig statt, d.h. der Client übernimmt die Anwendungs- und Repräsentationsaufgaben. Die Anwendungslogik kann somit vollkommen auf der Seite vom Browser, d.h. vom Client ausgeführt werden. Ein solches Programm, das innerhalb vom Browser ausgeführt wird, ist ein **Applet**. Eine besondere Rolle spielt hier die Programmiersprache `Java`. In `Java` geschriebene Programme können in einen maschinenunabhängigen Zwischencode übersetzt und Server-seitig abgelegt werden. Wenn eine `HTML`-Seite auf ein Java-Programm, ein sog. Applet verweist, dann wird es zum Client übertragen und dort mit einem entsprechenden Interpreter ausgeführt. So steht nichts mehr im Wege, auf dem Client Animationen ablaufen zu lassen, Eingabefelder zu überprüfen oder auch `Datenbank`-Anweisungen (`SQL`-) aufzurufen.
Einen geeigneten Satz von Funktionen für den Datenbankzugriff unter Java enthält die `JDBC` (Java Database Connectivity)-Schnittstelle. `JDBC` ist die Spezifikation einer Schnittstelle zwischen Client-Anwendung und `SQL`.
Die Bearbeitung von Daten einer Datenbank durch ein Java-Programm umfaßt im Rahmen der JDBC-Schnittstelle:

Zuerst ruft das Java-Programm die `getConnection()`-Methode auf, um das `Connection`-Objekt zu erstellen. Dann ergänzt es das `Statement`-Objekt und bereitet die SQL-Anweisungen vor. Ein SQL-Statement kann sofort ausgeführt werden (Statement-Objekt) oder ein kompilierbares Statement (Prepared-Statement-Objekt) oder ein Aufruf an eine gespeicherte Prozedur (Callable-Statement-Objekt) sein. Falls die `executeQuery()`-Methode ausgeführt wird, wird ein Resultat-Objekt zurückgeschickt. SQL-Anweisungen wie `update` und `delete` verwenden die `executeUpdate()`-Methode. Sie gibt einen ganzzahligen Wert zurück, der die Anzahl der Zeilen anzeigt, die vom SQL-Statement betroffen sind.
Der `ResultSet` enthält Zeilen mit Daten, die mit der Methode `next()` abgearbeitet werden können. Bei einer Anwendung mit Transaktionsverarbeitung können Methoden wie `rollback()` und `commit()` verwendet werden.

Zur Ausführung kann ein Java-Programm über mehrere JDBC-Treiber angesteuert werden. Jeder Treiber registriert sich beim JDBC-Manager währen der Initialisierung. Beim Versuch, sich mit einer Datenbank zu verbinden, gibt das Java-Programm einen Datenbank `URL` an den JDBC-Manager weiter. Der JDBC-Manager ruft dann einen der geladenen JDBC-Treiber auf. Die `URL`s von JDBC haben die Form "`jdbc:subprotocol:subname`". "`subprotocol`" ist der Name der jeweiligen Datenbank, z.B.: "`jdbc:oracle:oci7:@rfhs8012_ora3`"

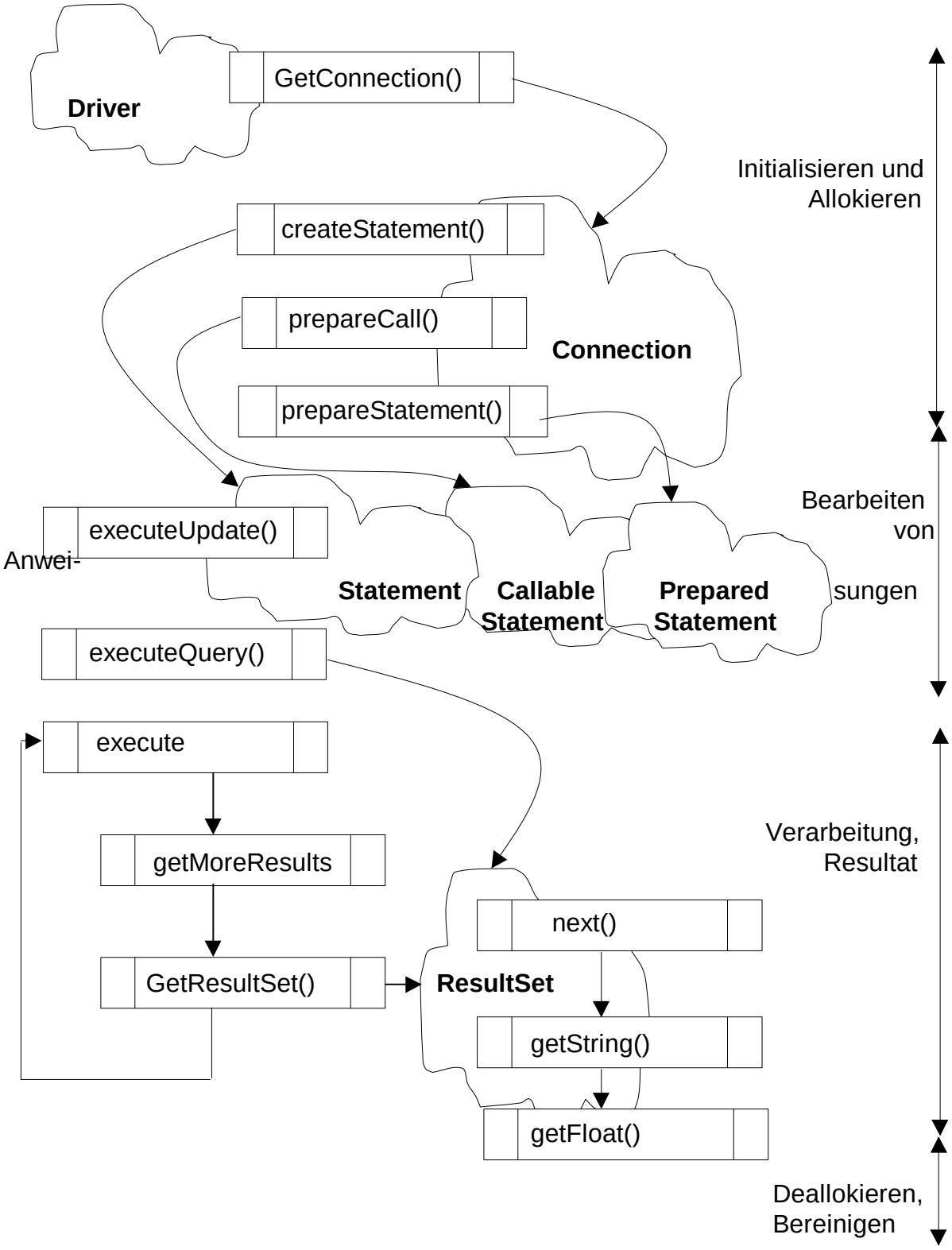


Abb. 1.5-20: JDBC-Objekte, Methoden, Ablauf

1.6 Verteilte Systeme

1.6.1 Verteilte Datenbanken

Definition und Anforderungen

"**Verteilte Datenbanken**" sind auf verschiedenen Rechnern eines Rechnernetzes verteilt und besitzen ein Datenbankverwaltungssystem (DBMS), das die Daten in einheitlicher Form verwaltet. Für den Benutzer eines derartigen Systems soll die physische Verteilung der Daten nicht sichtbar sein, d.h.: Das System der verteilten Datenbanken (VDBS) realisiert eine logisch zentrale Sicht der physisch dezentral abgespeicherten Daten. Die Information über die Datenverteilung wird im Systemkatalog gehalten.
Verteilte Datenbanksysteme (VDBS) lassen sich einteilen in homogene und heterogene Systeme. Ein VDBS ist homogen, falls Datenmodell und Software auf allen Rechnern einheitlich sind, andernfalls heterogen.
Verteilte Datenbanken werden in vielen Fällen der Organisationsstruktur eines Unternehmens besser gerecht als herkömmlich zentralisierte Datenspeicher, da Änderungen in der Datenverteilung keine Auswirkungen auf bestehende Anwendungen haben.
Die Verteilung auf mehrere DBMS (in der Regel auf verschiedenen, geographisch verteilten Rechnern) unterstützt auf der einen Seite das Lokalitätsverhalten vieler Anwendungen, ermöglicht andererseits auch die zentrale, logische Sicht der Daten.
Die wichtigsten Anforderungen, die Verteilte Datenbanksysteme „idealerweise“ erfüllen sollten, formulierte C.J. Date[88] in 12 „Regeln“:

1. Lokale Autonomie
Lokale Daten werden unabhängig von anderen Standorten verwaltet. Jeder Rechner sollte ein Maximum an Kontrolle über die auf ihm gespeicherten Daten ausüben. Der Zugriff auf diese Daten darf nicht von anderen Rechnern abhängen.
2. Keine zentralen Systemfunktionen
Zur Unterstützung einer hohen Autonomie der Knoten und Verfügbarkeit sollte die Datenbankverarbeitung nicht von zentralen Systemfunktionen abhängen. Solche Komponenten bilden außerdem einen potentiellen Leistungsengpaß.
3. Hohe Verfügbarkeit
Fehler im System (z.B. Rechnerausfall) oder Konfigurationsänderungen (Installation neuer Software und Hardware) unterbrechen nicht die Datenbankbearbeitung.
4. Ortstransparenz
Der physische Speicherort sollte für Benutzer verborgen bleiben. Der Datenbankzugriff darf sich vom Zugriff auf lokale Objekte nicht unterscheiden.
5. Fragmentierungstransparenz
Eine Relation (Tabelle) der Datenbank sollte verteilt auf mehrere Knoten gespeichert werden können. Die dazu zugrundeliegende (horizontale oder vertikale) Fragmentierung der Relation bleibt für den Datenbankbenutzer unsichtbar.
6. Replikationstransparenz
Die mehrfache Speicherung von Teilen der Datenbank auf unterschiedlichen Rechnern bleibt für den Benutzer unsichtbar. Die Wartung der Redundanz obliegt ausschließlich der DB-Software.
7. Verteilte Anfragebearbeitung

[88] C.J. Date: An Introduction to Database Systems, 5th Edition (chapter 23), Addison Wesley, 1990

Innerhalb einer DB-Operation (SQL-Anweisung) sollte die Möglichkeit bestehen, auf Daten mehrerer Rechner zuzugreifen. Zur effizienten Bearbeitung sind durch verteilte DBMS geeignete Techniken bereitzustellen (z.B. Query Optimierung)
8. Verteilte Transaktionsverarbeitung
Das DBMS hat die Transaktionseigenschaften auch bei der verteilten Bearbeitung einzuhalten. Dazu sollten geeignete Recovery- und Synchronisationstechniken bereitstehen.
9. Hardwareunabhängigkeit
Die Verarbeitung der Datenbank sollte auf verschiedenen Hardware-Plattformen möglich sein. Sämtliche Hardware-Eigenschaften bleiben dem Benutzer verborgen.
10. Betriebssystemunabhängigkeit
11. Netzwerkunabhängigkeit
Die verwendeten Kommunikationsprotokolle und -netzwerke haben keinen Einfluß auf die DB-Bearbeitung.
12. Datenbanksystemunabhängigkeit
Es muß möglich sein, unterschiedliche Datenbanksysteme auf den enzelnen Rechnern einzusetzen, solange sie eine einheitliche Benutzerschnittstelle (z.B. gemeinsame SQL-Version) anbieten.

Referenzarchitektur für verteilte Datenbanken

Sie beschreibt ein idealtypisches Modell[89] für konzeptionelle Schichten in verteilten Datenbanken:

Globales Schema
Es beschreibt Datenobjekte, ihre Integritätsbedingungen und Berechnungs-strukturen aus globaler, d.h. nicht verteilter Sicht.
Fragmentierungsschema
Es teilt eine globale Relation (Tabelle) in ein oder mehrere logische Teile (Fragmente) und verteilt diese auf geeignete Rechnerknoten. Fragmente des Fragmetierungsschemas müssen bzgl. Auf die ihnen zugeordneten globalen Relationen drei Bedingungen erfüllen:

1. Alle Daten der globalen Relation müssen in ihren Fragmenten vertreten sein
2. Jede globale Relation muß sich aus ihren Fragmenten vollständig rekonstuieren lassen.
3. Die Fragmente sollen überschneidungsfrei sein[90].

Allokierungsschema
Es beschreibt, auf welchen Datenbankknoten des verteilten Systems ein Fragment physikalisch gespeichert wird (1:1 oder 1:n Verhältnis). Im Falle der 1:1-Verteilung spricht man von einer Dispersion, andernfalls von einer Replikation der Daten. Die Replikation erlaubt die permanente Abbildung des Fragments.
Lokales Schema
Hier werden die physikalischen Abbilder in die Objekte und das Datenmodell des jeweiligen datenbanksystems umgesetzt. Das lokale Schema ist abhängig vom Typ des zugrundeliegenden Datenbanksystems. Die ersten drei Schichten sind unabhängig vom benutzten DBS.

[89] vgl.: Ceri, Stefano u. Pelagotti, Guiseppe: Distributed Databases, Principles & Systems, McGraw-Hill 1985
[90] Ausnahme: Bei vertikale Fragmentierung wird der Primärschlüssel redundant gespeichert, um die Rekonstruktion der globalen Relation zu gewährleisten.

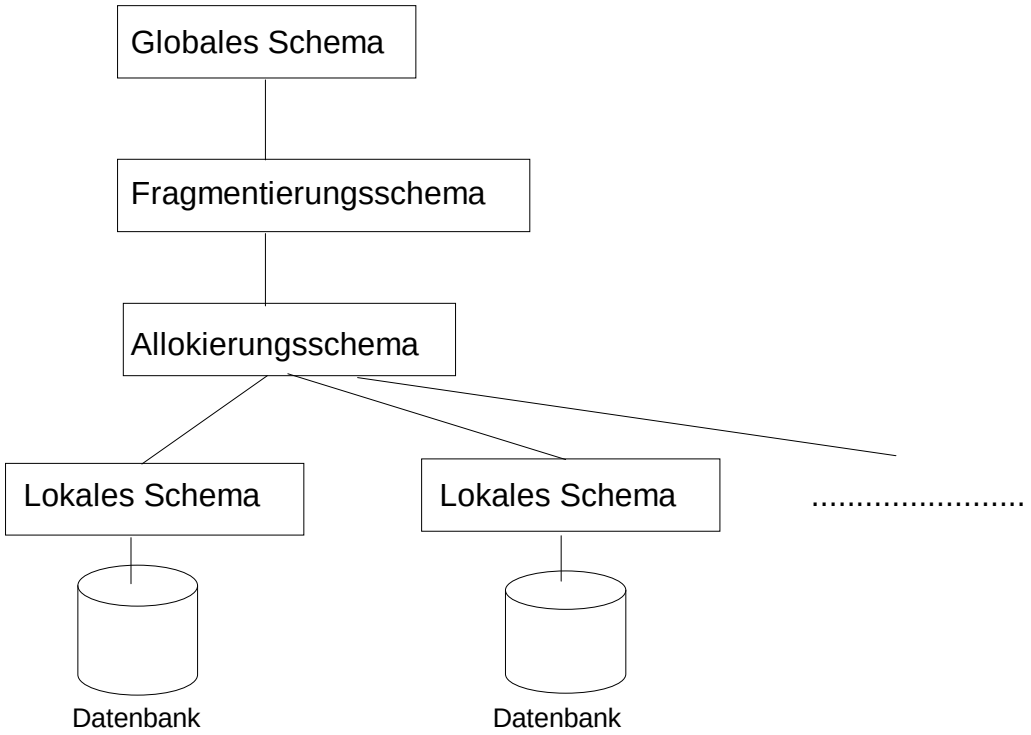


Abb. 1.5-21: Aufbau einer verteilten Datenbank

Rahmenbedingunen für das Design

Es gilt nicht nur wie bei zentralen Datenbanken das konzeptuelle Schema aufzubauen, es müssen auch Fragen der Fragmentierung, Allokierung bzw. Ortsverteilung gekärt werden. Wichtige Prinzipien für Verteilung und Replikation der Daten sind:

- Lokalitätsprinzip
  Daten sollen so nahe wie möglich an den Ort der Verarbeitung gelegt werden (Minimierung der in der Regel aufwendigen Zugriffe über das Netz).
- Optimierung der Verfügbarkeit und Zuverlässigkeit
  Durch geeignete Replikationen lassen sich die erhöhten Risiken komplexer Konfigurationen und Verwltung verteilter Datenbanken minimieren. Im Falle eines Systemabsturzes oder eines Netzausfalls kann die betroffene Applikation dann auf einen anderen Knoten mit replizierten Daten ausweichen.
- Optimierung der Lastverteilung durch Verteilung der Daten und der auf daten zugreifenden Applikationen unter Wahrung des Lokalitätsprinzips und größtmöglicher Parallelisierung der Verarbeitung.

Systemarchitektur

Physikalisch gesehen ist eine verteilte Datenbank ein über ein Telekommunikationssystem verbundenes Netz aus autonomen Rechnerknoten. Jeder Knoten verfügt über ein eigenes Datenbanksystem:
An die Datenbanksysteme der Rechnerknoten müssen sich sowohl lokale Abfragen ohne Berücksichtigung anderer Knoten absetzen und bearbeiten lassen als auch globale, mehrere Knoten einbeziehende Abfragen. Für diese Abfragen

gelten die Forderungen nach Verteilungs-, Replikations- und Fragmentierungestransparenz[91], d.h. jeder Rechnerknoten benötigt eine zusätzliche (globale) Komponete des **DBMS**, die diese Anforderungen erfüllt. Diese Komponente hat dann folgende Aufgaben:

- Bereitstellen von Wissen über die Verteilung der Daten (Speicherungsort, Replizierung, Fragmentierung mit netzweit eindeutiger Kennung). Die Bereitstellung dieser Metadaten übernimmt ein globales Datenwörterbuch.
- Erbringen der **ACID**-Eigenschaften im Rahmen einer globalen Transaktions-verwaltung, so daß ein Anwendungsprogramm glaubt, die Daten würden im **DBS** des Rechners gehalten, auf dem es abläuft.
- Lenkung der Zugriffe an die zuständigen Knoten im Rahmen der Auftrags-bearbeitung. Globale Anfragen werden in eine Reihe lokaler Anfragen aufgespalten, die an die betreffenden Knoten versandt werden und deren Antworten dann wieder eingesammelt und verknüpft werden. Die Auftragsbearbeitung erstellt dazu einen möglichst kostengünstigen Ausführungsplan (unter Ausnutzung der Kenntnisse von Speicherungsort, Replizierung, Fragmentierung und Übertragungskosten).

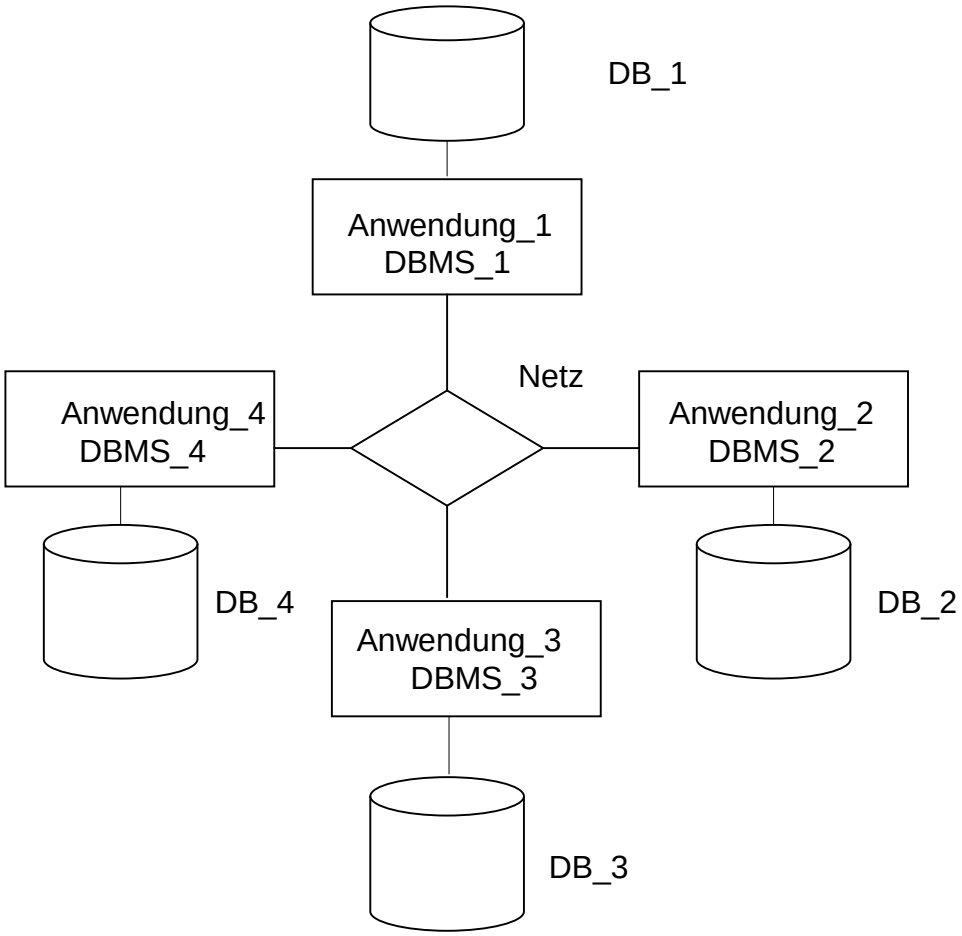


Abb. 1.5-22: Verteilte Datenbanken

Verteilungsstrategien

Verteilte Datenbanken können Verteilungen hinsichtlich der Daten, des Datenbankschemas, des Systemkatalogs, der Query-Auswertungen, der

[91] Die Anwendung arbeitet mit globalen Relationen, d.h. der Programmierer braucht keinen Kenntnis über Fragmentierung und Allokation.

Synchronisation und der Verfahren zur Fehlerbehandlung auf verschiedene Arten vornehmen. Das führt zu einer Fülle komplexer Probleme, die ein "Verteiltes Datenbanksystem" lösen muß.

Datenverteilung (Fragmentierung)

Die Fragmentierung ermöglicht es, Daten einer globalen Relation (Tabelle) auf unterschiedliche Knoten des verteilten Systems zu speichern. Abhängig von der Art, wie Tupel einer globalen Relation aufgeteilt werden, spricht man von horizontaler oder vertikaler Fragmentierung bzw. von einer Mischform der beiden.
Unter **horizontaler Fragmentierung** werden komplexe Tupel einer globalen Relation mit Hilfe der Selektion aufgeteilt. Die Relation (Tabelle) wird horizontal geschnitten.
Bsp.: Horizontale Fragmentierung der Relation [92] "Konto" nach Filialen. Jede Filiale speichert auf ihrem Rechner die sie betreffenden Daten.

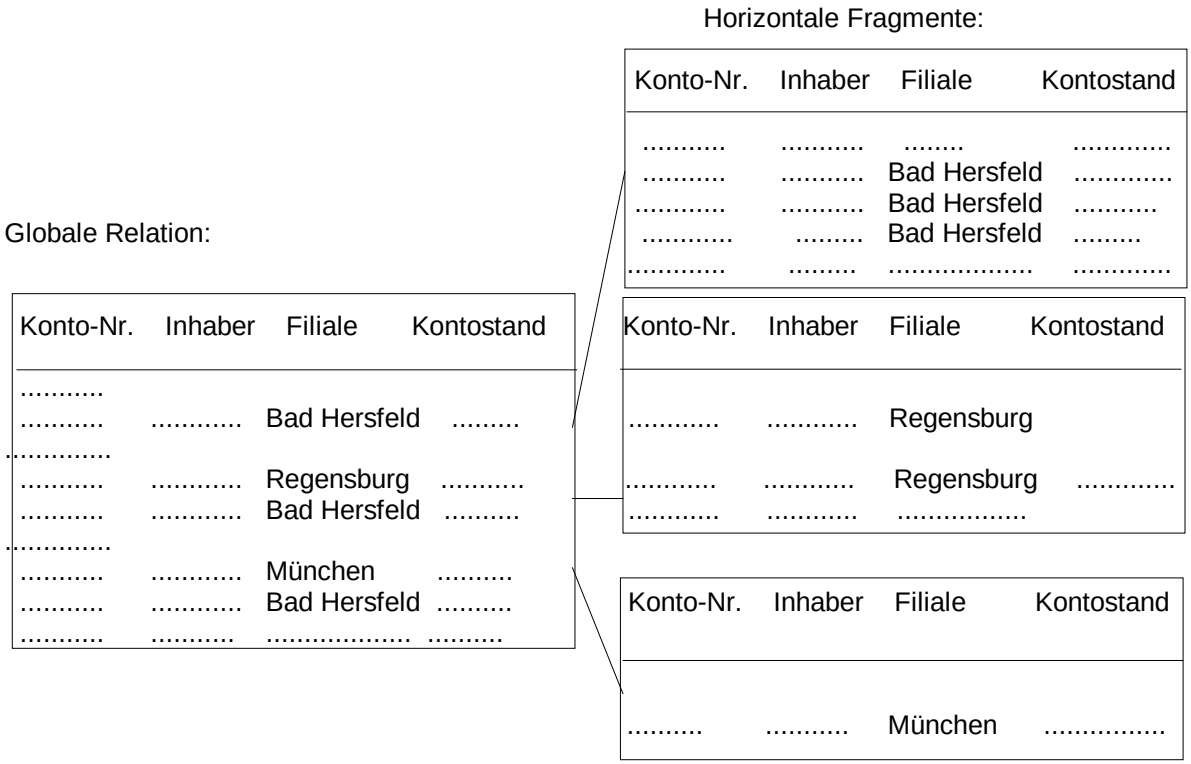


Abb. 1.5-23: Horizontale Fragmentierung der Kontorelation

Die **vertikale Fragmentierung** teilt die globale Relation in Abhängigkeit von den Attributen auf. Die Fragmentierung wird mit Hilfe der Projektion gebildet. Die Zuordnung der verteilt gespeicherten Attributgruppen erfolgt über den redundant abgebildeten Primärschlüssel.
Bsp.: Aufteilung der Relation "Artikel" in ein bestandsbezogenes und in ein preisbezogenes Tupel.

[92] In relationalen Datenbanken ist die Relation (Tabelle) Ausgangspunkt für die Verteilungseinheit

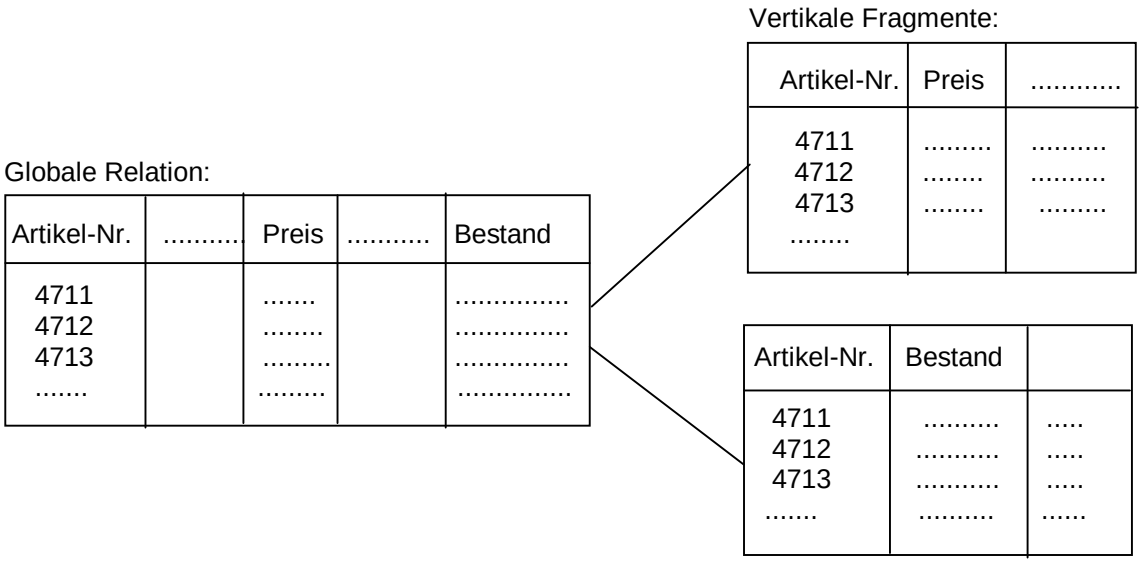


Abb. 1.5-24: Vertikale Fragmentierung der Realtion Artikel

Auch Mischformen sind denkbar. Dabei läßt sich bspw. ein vertikal gebildetes Fragment nochmals in eine Gruppe von horizontal gebildeten Fragmenten unterteilen.
Auf dem Fragmetierungsschema baut das Allakationsschema auf. Hier ist eine Entscheidung über Dispersion und Replikation der Fragmente zu treffen. Bei der Dispersion werden das Fragment einer globalen Relation genau einmal auf einen Knoten des verteilten Systems als Tabelle abgebildet. Der Zugriff auf die im Fragment gespeicherten Daten hat auf genau diesen Knoten zu erfolgen und ist damit abhängig von der Verfügbarkeit des betreffenden Knotens und der Verfügbarkeit des Netzwerks zur Erreichung des Knotens. Bei der Replikation werden Fragmente redundant gespeichert. Der Verbrauch an Speicherplatz steigt, auf die Daten kann aus mehreren Lokalitäten "näher" und "performanter" zugegriffen werden.

Katalogorganisation

Bei einem nicht voll redundanten VDBS muß zur Beantwortung der Anfrage (query processing) bestimmt werden, an welchem Knoten die angesprochenen Daten gespeichert sind. Informationen über Heimatadressen befinden sich im Systemkatalog (data dictionary, data directory). Für die Abspeicherung und Verwaltung der Katalogdaten sind ähnliche Probleme zu lösen wie bei den Benutzerdaten. Wichtige Formen der Katalogorganisation sind:

- zentralisierter Katalog
  Ein Rechner hält den vollständigen Katalog. Das Gesamtsystem ist allerdings von Leisung und Ausfallsicherheit eines Rechners abhängig
- Voll redundanter Katalog
  Jeder Rechner im verteilten System hält den Gesamtkatalog
- Lokaler Katalog
  Jeder Rechner hält einen Katalog über die lokale Dateien. Für verteilte Transaktionen entstehen Übertragungskosten, da die Adresse von Fremddateien nicht bekannt sind.

Schemaverteilung

Eine Datenbank wird bekanntlich durch das Datenbankschema beschrieben. Dort erfolgt die Definition

- der logischen Datenstrukturen, z.B. Relationen mit zugehörigen Attributen und Beziehungen zwischen den Relationen
- der physikalischen Strukturen z.B. Speicherstrukturen, Zugriffspfade

Ein **VDBS** benötigt zusätzlich die Definition der Verteilungsregeln, die einmal die Einheiten der Datenverteilung festlegen und zum anderen die Zuordnung der Dateneinheiten zu den Rechnern an unterschiedlichen Orten.
Verteilungsstrategien zur Schemaverteilung können unter dem Aspekt "Zentralisierung" bzw. "Verteilung" betrachtet werden. Es bestehen folgende Möglichkeiten, das Schema in die Verteilung mit einzubeziehen:

1. Nicht redundante Strategien

Zentralisierung: Die vollständige Schemabeschreibung befindet sich in einem Knoten. Jeder Verkehr mit der Datenbank impliziert einen Zugriff auf diesen ausgezeichneten Knoten.
Verteilung: An jedem Ort des Rechnerverbunds sind neben den lokalen Daten (deren Verteilung soll ebenfalls nicht redundant gespeichert sein) auch die zugehörigen Teile des Schema gespeichert. Ausschließlich lokale Verarbeitung benötigt keinen Netzzugriff mehr. Bei jeder nicht lokalen Anfrage (Query) muß in allen Knoten geprüft werden, ob sie die gewünschten Daten bereitstellen können.

2. Redundante Strategien

Zentralisierung: Jeder Knoten enthält die Schemainformation für seine Daten, zusätzlich steht aber das gesamte Schema in einer Zentrale zur Verfügung. Nicht lokale Zugriffe wenden sich zunächst an die Zentrale, um dort den Ort der gewünschten Daten zu erfahren.
Verteilung: Jeder Rechner kennt das vollständige Schema. Änderungen des Schemas sind an allen Rechnern gleichzeitig vorzunehmen. Benutzeraufträge können am Ort ihrer Eingabe vollständig gestartet und verwaltet werden.
Kombinationsmöglichkeiten: Irgendeine Teilmenge des Schemas kann als Verteilungseinheit herangezogen werden. Das bringt große Flexibilität, erzwingt den Aufbau eines "Schemas für das Schema (directory directory) zur Festlegung, welche Teile wo gespeichert sind.

Query Processing

Zwei wesentliche Unterschiede kenn zeichnen die Bearbeitung von Datenbankabfragen (Queries) in einem **VDBS** gegenüber einem zentralen **DBS**:

1. Zusätzliche Verzögerungen durch Netzkommunikation
2. Nutzung von Parallelität durch Beteiligung mehrerer Rechner und redundant vorhandener Daten

Bekannt sind folgende "query processing"-Strategien:

- Zerlegen der Anfrage an zentraler Stelle so, daß jede Teilabfrage von genau einem Knoten dezentral bearbeitet werden kann. Das Formuliern der zu verschickenden Teilabfragen erfolgt in der Abfragesprache des Datenbanksystems (DML), z.B. in SQL (Versenden von DML-Anfragen zwischen Programmen).
- Ausarbeiten eines Gesamtplans zur Bearbeitung der Anfrage
  Der Ausführungsplan umfaßt:
  -- Zerlegung der kompexen Anfragen in Teilfragen
  -- Festlegen der Ausführungsreihenfolge dieser Teilfragen (sequentiell, parallel,..)
  -- Bestimmen der Knoten an die Daten geschickt werden sollen
  -- Auswahl lokaler Zufriffspfade
  -- Optimierung
  Hier ist festzulegen, wie lokale Bearbeitung und Übertragung von Teilergebnissen aufeinander folgen bzw. sich abwechseln sollen

Zielsetzung einer Anfrageübersetzung ist es, zu einer Anfrage einen Ausführungsplan festzulegen, der eine möglichst effiziente Bearbeitung des Auftrags ermöglicht. Diese Aufgabe ist bereits für zentralisierte DBS komplex, da im allg. eine große Anzahl alternativer Ausführungsstrategien besteht. Noch komplexer ist die Bestimmung eines optimalen Ausführungsplans im verteilten Fall.

Synchronisation

Falls mehrere Transaktionen (Prozese) in DBS gleichzeitig diesselben Daten benutzen, werden Synchronisationsmaßnamen erforderlich. Die parallele (nebenläufige, "concurrent") Ausführung muß dasselbe Ergebnis erzielen wie irgendeine sequentielle Reihenfolge. In einem VDBS gibt es darüber hinaus zusätzliche Probleme, bedingt bspw. durch die

- Existenz mehrfacher Kopien
- mögliche Beteiligung mehrerer Rechner an der Bearbeitung einer Transaktion

Verfahren zur Fehlerbehandlung (crash recovery)

Aufgabe der Verfahren ist es, bei Störungen

- die Datenbankkonsistenz zu erhalten (bzw. wieder herzustellen)
- Die Serialisierbarkeit aller benachbarten Transaktionen zu gewährleisten
- die Auswirkungen auf die direkt betroffenen Transaktionen zu beschränken

Die Basis der Fehlerbehandlungsverfahren bilden lokale Sicherungsverfahren in den Knoten des VDBS. Derartige Knoten verfügen über ein solides, sicheres Hintergrundsystem, dessen Inhalt alle Störungen überlebt. Lokale Sicherheit bietet die Grundlage für die Implementierung im globalen Fehlerbehandlungsverfahren.
Eine wesentliche Voraussetzung für die Bewahrung der Konsistenz in "verteilten DB" ist: Die Erhaltung der Unteilbarkeit globaler Änderungstransaktionen.
Bsp.: Eine Transaktion T hat in den Knoten K_i .. K_m über Teiltransaktionen T_i ... T_m neue Versionen erzeugt. Diese Versionen müssen in allen oder in keinem Knoten gültig gemacht werden. Es muß sichergestellt werden, daß sich etwa aufgrund eines Knotenzusammenbruchs eine Teiltransaktion zurückgesetzt, eine

andere jedoch richtig beendet wird. Innerhalb einer einzelnen Transaktion T_i von Knoten K_i hat die zugehörige lokale Fehlerbehandlungsroutine für die Konsistenz der Daten gesorgt.

1.6.2 Datenbankrechner und Datenbankmaschinen

Hohe Softwarekosten und immer preiswertere Hardware haben dazu geführt, das Konzept des Universalrechners zu überdenken. Man kann heute einen Trend zur Aufteilung veschiedener Systemkomponenten auf autarke Prozessoren beobachten. Für Datenbanken sind zwei Enwicklungsrichtungen festzustellen:

1. Entwicklung neuer Rechnerfamilien, die sowohl assoziatives (inhaltsorientiertes) als auch paralleles Arbeiten ermöglichen. Diese Rechner führen die Suche nach Antwortmengen nicht mehr über physische Adressen aus, sondern ermöglichen das Wiederauffinden der Daten inhaltsmäßig über Assoziativspeicher.
2. Auf konventionellen Rechnerarchitekturen beruhende autarke Prozessoren

Die Konsequenz aus der 1. Entwicklungsrichtung führt zu spezieller Such- und Verarbeitungshardware und diejenige aus der 2. Entwicklungsrichtung zu dem Konzept der Datenbankrechner. Beide Trends führen zur Datenbankmaschine.

Ein **Datenbankrechner** ist ein Rechensystem, das ausschließlich Datenbankaufgaben wahrnimmt. Die Kommunikation zu einem Verarbeitungsrechner muß gewährleistet sein.
Eine **Datenbankmaschine**[93] ist ein Datenbankrechner, der für die Durchführung seiner Datenbankaufgabe durch spezielle, diese Aufgabe unterstützende Hardware ausgerüstet ist. Neben einem intelligenten Kontroller, der eine schnelle Reduktion der Gesamtmenge auf eine komprimierte Zwischenergebnismenge durchführt, sollte eine Datenbankmaschine auch hardwaregestützte Operationen (z.B. zur Bildung von invertierten Listen, Berechnen von Hashschlüsseln bzw. Sortieren und Verknüpfen von Tupelmengen) umfassen. Diese Operationen können leistungsmäßig entweder durch hohe Parallelität von vielen relativ einfachen Hardwarebausteinen (z.B. Mikroprozessoren) oder von wenigen speziell entwickelten Hochleistungsbausteinen durchgeführt werden.

1.7 Integritätsbedingungen und Integritätsregeln

Konsistenz oder Integrität bedeutet die Korrektheit der in der Datenbank gespeicherten Daten. Dazu gehört die technische oder physische Integrität, d.h. richtiges Arbeiten des Datenbanksystems. Darüber hinaus sollen die Daten aber weitgehend den Abhängigkeiten und Bedingungen genügen, denen die durch die Daten dargestellten realen Objekte unterworfen sind ("logische Konsistenz").

[93] vgl.: Eberhard, L. und andere: Datenbankmaschinen - Überblick über den derzeitigen Stand der Entwicklung. Informatik-Spektrum, Heft 4, S. 31 - 39, (1981)

1.7.1 Integritätsbedingungen

Sie können auf verschiedene Weise klassifiziert werden.

Physische Integrität

Darunter versteht man neben der physischen Datenkonsistenz

- die Vollständigkeit der Zugriffspfade
- die Vollständigkeit der Speicherstrukturen
- die Vollständigkeit der Beschreibungsinformationen

Semantische Integritätsbedingungen

Ganz allgemein versteht man unter semantischer Integrität die Übereinstimmung von realen und gespeicherten Daten. Speziell könnte man hier unterscheiden:

- Bereichsintegrität
  Sie umfaßt den Erhalt der Attributwerte innerhalb der Relationen. Jedes Attribut einer Relation hat einen bestimmten Wertebereich.
- Intra-relationale Integrität
  Sie umfaßt die Korrektheit der Beziehungen zwischen den Attributen in einer Relation (z.B. funktionale Abhängigkeiten). Eine der wichtigsten Integritätsreglen ist in diesem Zusammenhang: Die Eindeutigkeit des Schlüssels. Schlüsseleindeutigkeit bedeutet: Kein neues Tupel kann eingetragen werden, falls Schlüsselwerte mit Werten des bestehenden Tupels in dieser Realtion übereinstimmen.
- Entitätsintegrität
  Sie verbietet Null-Werte im Feld des Primärschlüssels und garantiert, daß Sätze erst dann in die Datenbank aufgenommen werden, wenn das wichtigste Attribut (der Satzschlüssel) vorhanden ist.
- Beziehungsintegrität (referentielle Integrität)
  Sie sorgt in relationalen Datenbanken dafür, daß jedem Wert eines Fremdschlüssels ein entsprechender Primärschlüssel des gleichen Bereichs zur Verfügung steht.

Idealerweise speichern Datenbankverwaltungssysteme (**DBMS**), die Integritätskonzepte unterstützen, Integritätsbedingungen (z.B. zulässige Wertebereiche für Daten) im **Data Dictionary**.

Pragmatische Integritätsbedingungen

Dazu zählen gesetzliche und organisatorische Vorschriften.

Ablaufintegrität

Hierunter versteht man die Datenkonsistenz bei der Verwaltung des Mehrbenutzer-betriebes. Die Ablaufintegrität kann beeinträchtigt werden, wenn mehrere Benutzer zugleich auf eine Datenbank zugreifen.
Falls eine Verletzung der Integrität festgestellt wurde, so muß sie wiederhergestellt werden. Dies geschieht durch einen Wiederherstellungsprozeß, der von der Art des Fehlers abhängig ist.
So verfügt das Datenbankverwaltungssystem in der Regel über eine transaktionsorientierte Fehlerbehebung. Eine Transaktion ist bekanntlich eine Folge von

Operationen, die alle insgesamt abgeschlossen sein müssen. Zu jeder Transaktion wird ein Protokolleintrag (Anfangszeitpunkt der Transaktion, Transaktionsnummer, Transaktionstyp, Datenobjekt vor Veränderung (before image), Zustand des Datenobjekts nach Veränderung (after image), Fehlerstatus, Endzeitpunkt der Transaktion) erstellt. Im Fehlerfall besteht die Möglichkeit fehlerfreie von fehlerhaften Transaktionen aus dem Protokoll zu rekonstruieren. Fehlerhafte Transaktionen können dann (in umgekehrter Reihenfolge) zurückgesetzt werden.

1.7.2 Integritätsregeln

Sie gewährleisten die logische Korrektheit (Widerspruchsfreiheit) der Datenbank und sorgen damit für die Einhaltung der **Integritätsbedingungen**. Sie können in Datenbanksystemen (z.B. in Oracle V 7) als **Trigger** oder **Constraints** implementiert werden. Constraints beschreiben Integritätsregeln im Rahmen der Datendefinitionen bei der Erstellung des Datenbank-Schema. **Trigger** sind Programme, die komplexe Fälle, die nicht mit **Constraints** gelöst werden können, beschreiben. In beiden Fällen erfolgt die Ausführung einer Integritätsregel ereignisorientiert, d.h. Das Ändern, Einfügen oder Löschen eines Datenelements führt zur Ausführung einer Integritätsregel.